DESIGN OF CROSSBAR ARCHITECTURE FOR VECTOR PROCESSING

A Thesis
Presented to
The Academic Faculty

By

Soundarya Bhagi

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2019

DESIGN OF CROSSBAR ARCHITECTURE FOR VECTOR PROCESSING

Approved by:

Dr. Arijit Raychowdhury, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology
Dr. Shimeng Yu
School of Electrical and Computer Engineering
Georgia Institute of Technology
Dr. Asif Khan
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 5, 2019

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

ACKNOWLEDGEMENTS

First and foremost I would like to express my sincere gratitude to my thesis advisor, Dr. Arijit Raychowdhury for his patient and focused guidance to me through each stage of the process. I started as a graduate student under him in Fall 2018 and worked with him on a special problem. His knowledge and expertise in memory design inspired me to take up thesis in the second semester of my Masters program and I haven't looked back ever since. He encouraged me to push my limits and learn new things and I cannot thank him enough for this. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

A very special gratitude to a few of his PhD students - Brian Crafton, Muya Chang and Insik Yoon - who constantly inspired me to make necessary changes for the betterment of my work. Their guidance and criticism at important stages motivated me to successfully complete my research and obtain fruitful results.

I would also like to thank ICSRL, Georgia Institute of Technology for providing me with the resources I needed to conduct simulations for my research. I would like to appreciate the efforts of everyone at the ECE Graduate Office at Georgia Institute of Technology for their help in all the administrative matters. I am also grateful to CBRIC: Center for Brain-Inspired Computing Enabling Autonomous Intelligence for funding my research.

Finally, I would like to thank my parents and my brother for providing me with moral and emotional support and motivating me throughout my Masters program and research during my thesis.

Thanks for all your encouragement!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This research aims at modeling the effect of R_{off} to R_{on} ratio for a binary Resistive Random Access Memory (RRAM) based crossbar architecture with specific focus on deep learning application such as image classification. The crossbar structure uses emerging non-volatile memory (eNVM) array architecture and is simulated with complex neural networks to obtain metrics such as accuracy, inference and run-time. Model validation is performed by running benchmark image datasets. It will be possible to obtain other hardware results when this project is implemented on actual hardware.

CHAPTER 1

INTRODUCTION AND BACKGROUND

The debate on ”memory wall” problem has been going on for a long time now. Many new ideas have since then been conceived and implemented. The memory wall concept theorized by Wulf and McKee in 1994[1], revolves around the idea that computer processing units (CPUs) are advancing at a pace that will leave random access memory (RAM) stagnant and the performance gap between processor and memory will keeping on widening as shown in Figure 1.1.

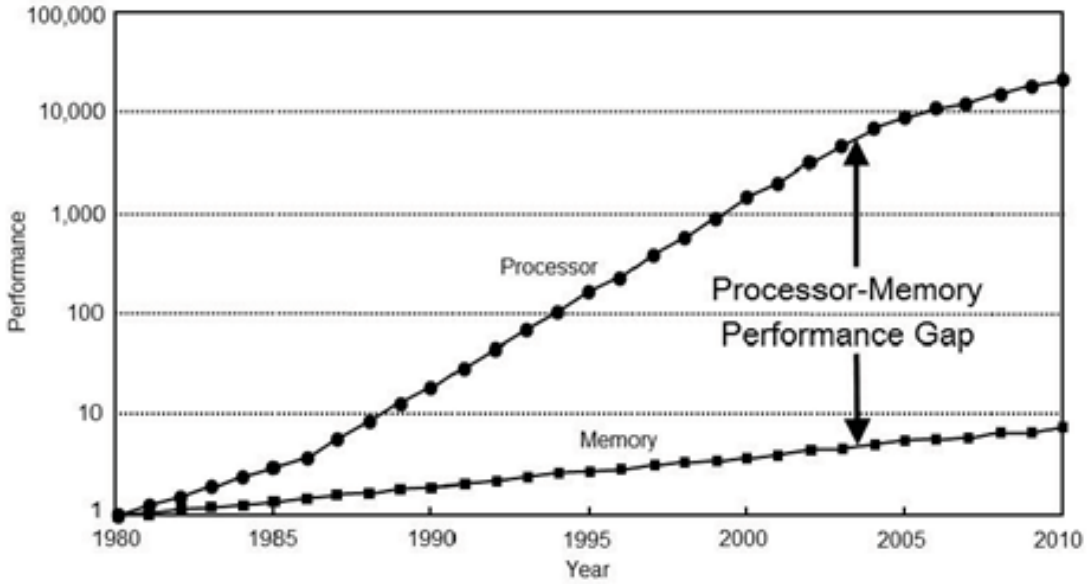


Figure 1.1: Processor-Memory Performance Gap. Taken from: Hennessy, et al.[2], Figure 5.2

As reviewed by An Chen[3], high-performance and low-cost emerging non-volatile memories (NVMs) simplify memory hierarchy, introduce non-volatility in logic gates and circuits, reduce system power, and enable novel architectures whereas storage-class mem-

ory (SCM) based on high-density NVMs such as resistive random access memory (RRAM or ReRAM), Phase Change Memory (PCM), conductive-bridging RAM (CBRAM) 3D X-point could fill the performance and density gap between memory and storage.

Special emphasis is being given nowadays to the use of RRAM crossbars for performing parallel computations and neural networks. New approaches using memristor and RRAM based crossbars have been reported in literature to enable processing of huge amounts of data. Some of the earlier reported works include design of a framework for evaluation of Deep Neural Networks (DNNs) on resistive crossbars[4], optimization of framework for AI applications[5], neuromorphic computing using RRAM crossbars[6]-[7], energy scaling advantages of crossbars[8], high performance and simple fabrication etc. Some of these designs even give high throughput at low energy and area consumption[9].

The idea of mapping the weight matrix of the neuron layers onto a memristor crossbar is not new. Tarkov [10] carried out simulations of adaptive adder with memristor synapses in the LTspice and proposed a method of mapping for image recognition application. Various hardware accelerators based on crossbar architecture have been proposed and some have reconfigurable computing architecture. Zidan et al.[11] described one such concept called Field-Programmable Crossbar Array (FPCA).

Hardware accelerators are specialized computer hardware for efficient and high performance computation of machine learning and deep learning applications. Use of SRAM and NVMs to model synaptic weights is constantly being explored to find a promising approach to increase computational efficiency in high performance hardware representation of neural networks.

Previous works reported in literature do not account for the the change in accuracy and inference values due to the OFF/ON resistance ratio ($\frac{R_{off}}{R_{on}}$). This project aims to find how much the OFF/ON resistance ratio can affect the accuracy results when used with neural networks and also identify the optimal OFF/ON resistance ratio to get high test accuracy of

given data.

1.1 Crossbar Array Architecture

Memristors are non-volatile electronic memory devices whose resistance can be programmed to high-resistance-state (HRS) or low-resistance-state (LRS) depending on a given condition. LRS allows more current to flow and thus can model a binary 1 whereas HRS restricts the flow of current and models a binary 0.

Thus memristors[12]-[13] are promising building blocks for upcoming emerging NVM[14] and artificial neural networks[15]-[18]. Organizing small memristors into high-density crossbar arrays is critical to meet the ever-growing demands in high-capacity and low-energy consumption.

RRAM operates by changing the resistance of a specially formulated solid dielectric material. The resistance state is changed based on voltage applied. RRAMs have many applications due to their robustness. Apart from storage, they are also used for automotive and internet of things (IoT) applications. RRAMs are also useful for modeling the synaptic weights of neurological synapses and implementing neural network architectures [19]-[21]. RRAMs can serve as standalone storage-class memory with 3D cross-point array in the long run.

Memristors are used in RRAM structure as the resistive element to model any particular cell to either allow current to pass through or inhibit current flow. In the crossbar array structure of this project, the output voltage for each bit-line is calculated using the equations derived by Hu, Miao & Li, et al.[22].

Figure 1.2 shows N X N memristor crossbar structure where a memristor is connected between each pair of horizontal word-line (WL) and vertical bit-line (BL).

Let V_I denote a vector of input voltages on WLs. Current at each BL can be obtained by measuring the voltage across a resistor with conductance g_s. If the memristor at the connection between WL_i and BL_j has a conductance of $g_{i,j}$ then the output voltage on the

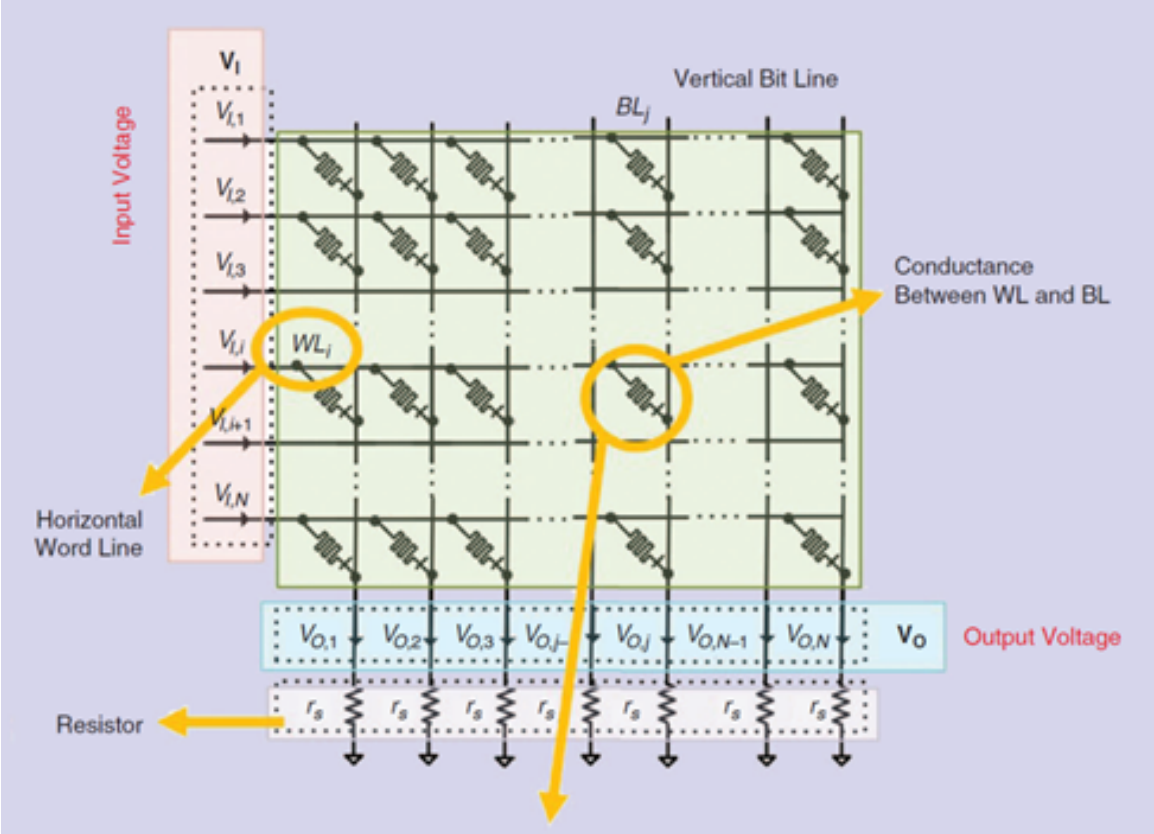


Figure 1.2: N x N Memristor Crossbar. Taken from: S. Liu, et al.[21], Figure 2

jth BL $V_{O,j}$ is given by:

$$V_{O,j} = [\frac{g_{1,j}}{\sum_{i=1}^{N} g_{ij}} .... \frac{g_{N,j}}{g_s + \sum_{i=1}^{N} g_{ij}}]V_I \tag{1.1}$$

which can be written as

$$V_O = CV_I \tag{1.2}$$

$$C = diag(\{\frac{1}{g_s + \sum_{i=1}^{N} g_{ij}}\}_{j=1}^{N})G^T \tag{1.3}$$

where $diag(\{x_i\}_{i=1}^{N})$ denotes a diagonal matrix with diagonal entries $x_1, x_2,..., x_N$, and G is the conductance matrix of memristors whose (i, j)th entry is given by $g_{i,j}$.

1.2 Neural Networks

Neural Networks (NN) are computing systems modeled after the neural network formed by neurons of a human brain that are designed to recognize patterns. Neural networks are a part of Artificial Intelligence (AI) along with various technologies like deep learning and machine learning.

McCulloch–Pitts neuron model is the basis for theoretical formulations of neural activity which influenced a variety of fields such as including neurosciences, computer science, artificial neural networks, and artificial intelligence[23].

A simple artificial neuron of the McCulloch-Pitts model is shown in Figure 1.3.

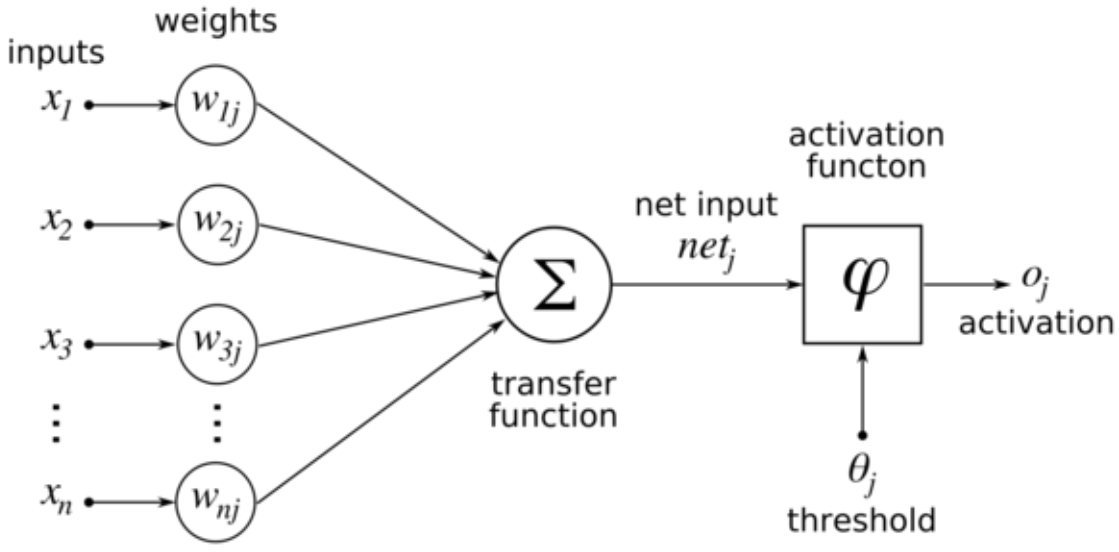


Figure 1.3: McCulloch-Pitts Model. Taken from: De Oliveira, et al.[24], Figure 2

Here, the inputs $x_1, x_2, \ldots\ x_n$ are weighted based on the synaptic weights $w_{1j}, w_{1j}, \ldots$ w_{nj} and the output Y is calculated based on threshold activation (or bias) of neuron θ and activation function ϕ according to the following equation:

$$Y = \phi(\sum w_i x_i + \theta) \tag{1.4}$$

Neural networks are constructed from 3 types of layers namely input layer, hidden layers and output layer. Input layer consists of the initial input data of the neural network. Hidden layers are the intermediate layers between input and output layer and this is where all the computations are done. Output layer is where the final results are produced for the given input data. Figure 1.4 shows a simple neural network with 1 input layer, 2 hidden layers and a output layer.

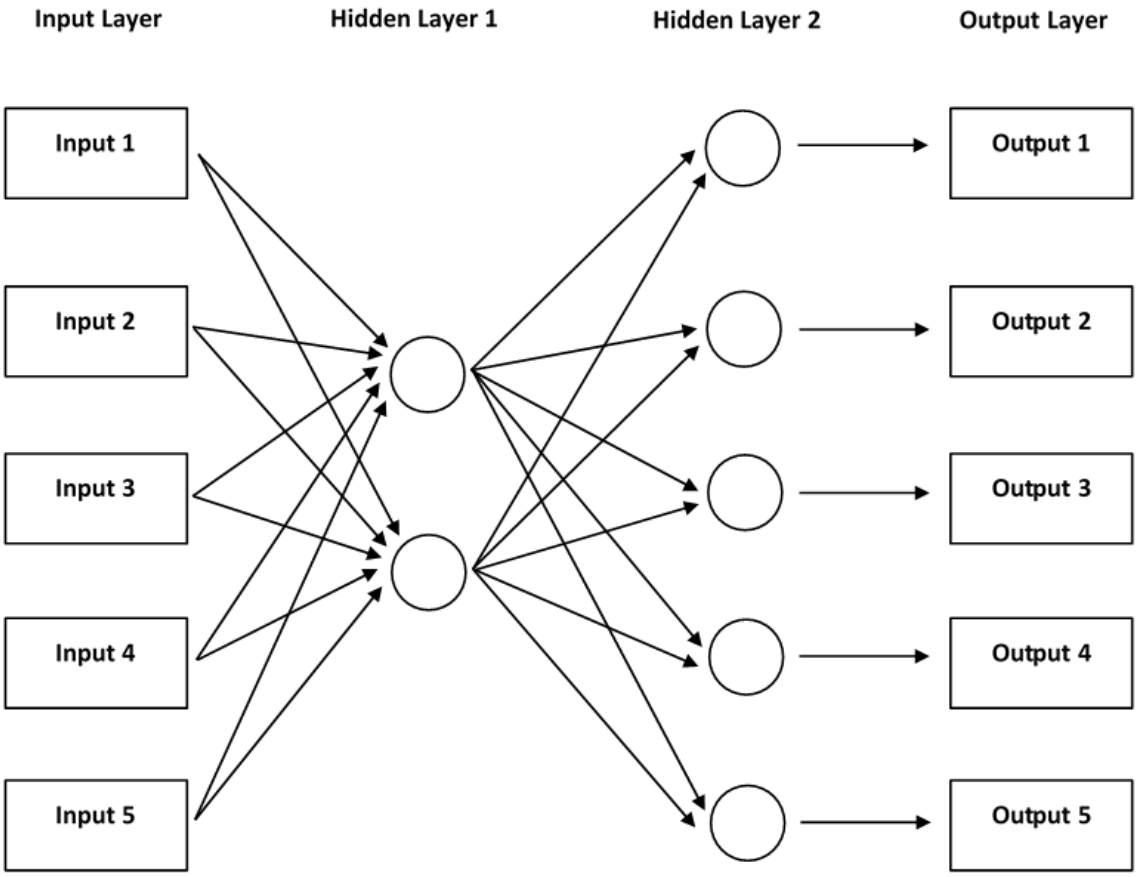


Figure 1.4: Simple Neural Network

Neural networks are trained with huge datasets. Once the neural network has been trained on samples of data, it can make predictions by detecting similar patterns in future data. The accuracy obtained during training of data is called training accuracy while the accuracy obtained when this trained network is tested with new data is called test accuracy.

In 2012, Alex Krizhevsky[25] used neural networks to win ImageNet competition (ILSVRC2012) and that is when neural networks grew to prominence. Deep Learning is becoming popular due to its high performance across many types of data. Deep learning can be used to build a convolutional neural network (CNN) to classify images. The importance of CNN has increased in the recent years and the advanced improvements of CNN on different aspects, including layer design, activation function, loss function, regularization, optimization and fast computation as discussed by Gu, et al.[26] along with their various applications in computer vision, speech and natural language processing.

The main idea of a CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we have to design an architecture which is not only good at learning features but is also scalable to massive datasets[27].

1.3 Crossbar Infrastructure in Neural Networks

Emerging non-volatile memory array architectures such as RRAM are being explored for data-intensive applications like in hardware accelerators where data is continuously transferred between logic and memory. Ni, Huang, & Liu et al.[28] explored distributed in-memory accelerator on binary RRAM crossbar for machine learning and improved computational energy efficiency and robustness by a binary RRAM crossbar for memory and logic units.

Figure 1.5 shows a traditional analog-fashion RRAM crossbar with Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC).

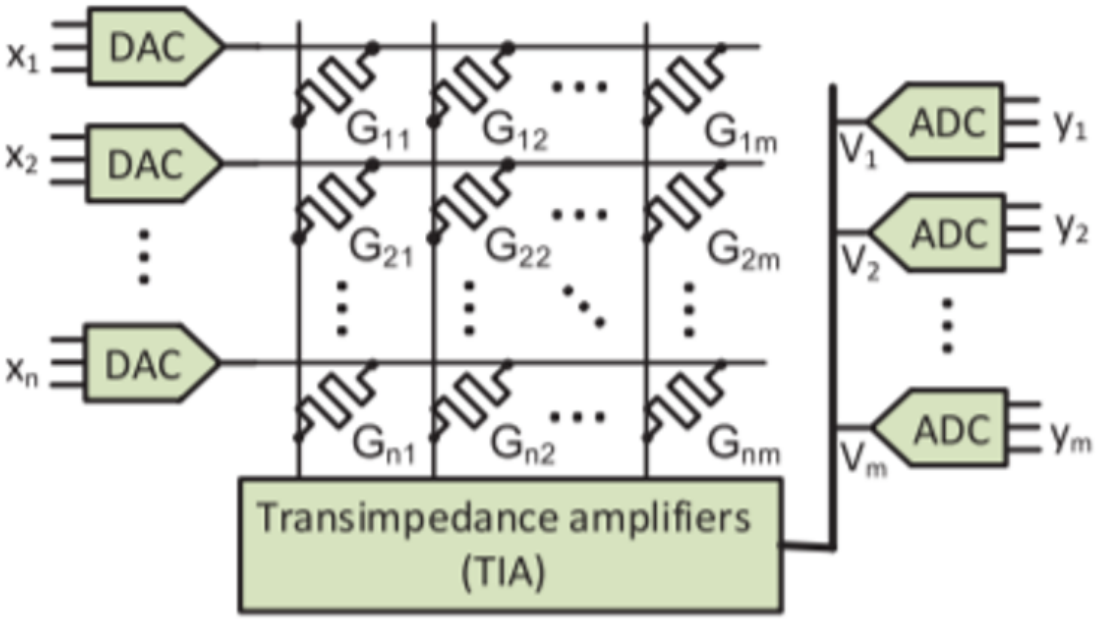


Figure 1.5: RRAM Crossbar with ADC and DAC. Taken from: Ni, et al.[28], Figure 4

The input and output interface of logic crossbar requires ADC to converter digital input given by user to analog voltage input which will be fed to the crossbar and DAC to convert analog signals back to digital for further processing by user.

In-memory computing of data in accelerators is also being explored in the matrix-vector multiplication on binary RRAM crossbars. Concepts where significant speedup can be achieved for matrix-vector multiplication in neural network–based machine learning, in addition to large energy savings when compared to the traditional CMOS-based out-of-memory computing architecture such as [28], are being explored.

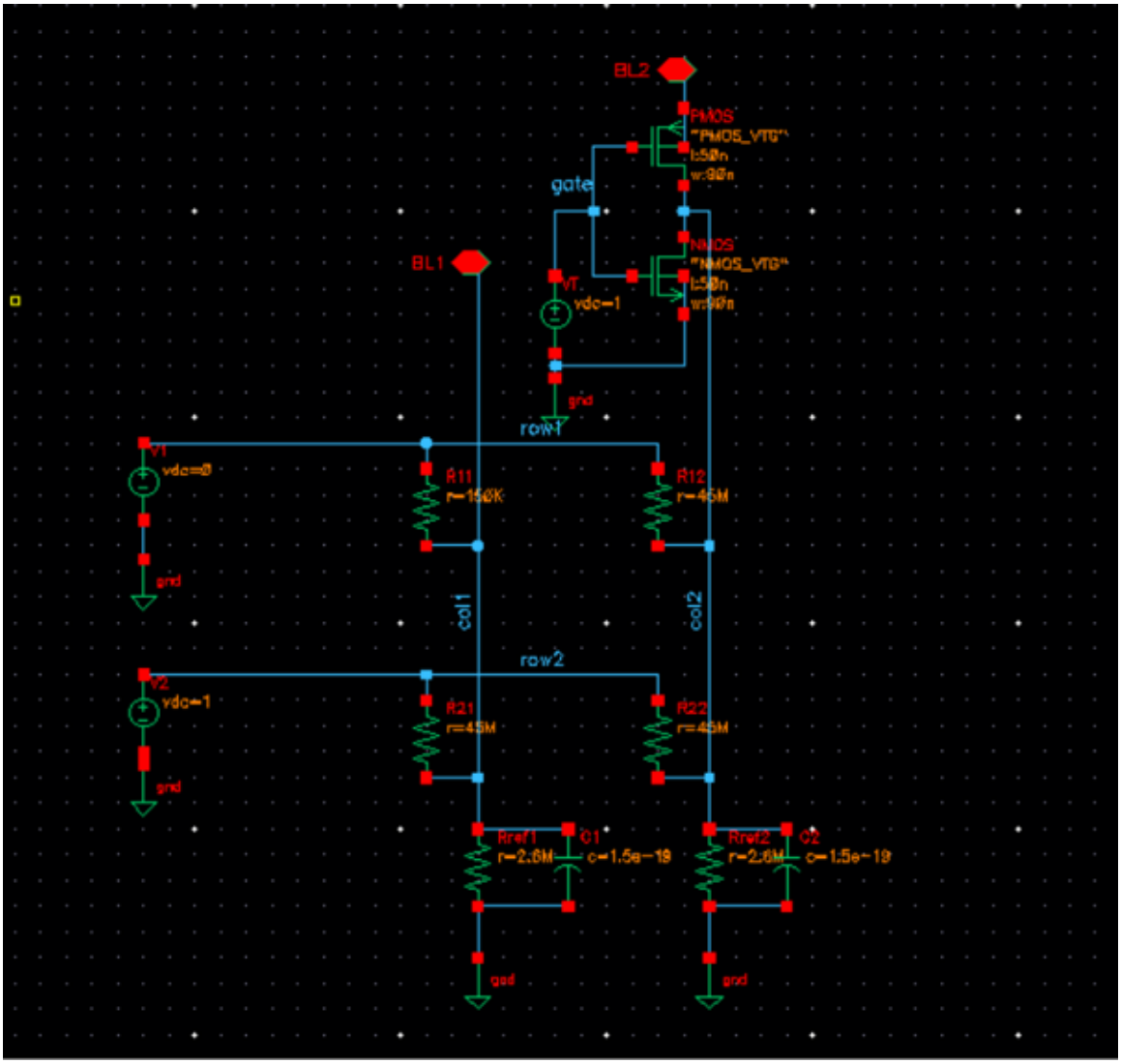


Figure 1.6: Schematic showing a 2 $\times$ 2 Crossbar Structure in Cadence Virtuoso

In this research, the structure of the crossbar was first designed in Cadence Virtuoso as a schematic and simulated using Cadence Spectre Circuit Simulator to obtain output voltage, current and noise values. Figure 1.6 shows the schematic of a 2 $\times$ 2 crossbar designed in Cadence Virtuoso. The results obtained by performing noise analysis on this crossbar have been provided in Chapter 4 with further explanation.

This hardware structure was then implemented on MATLAB to calculate the ideal output voltage using the equations (1.1) to (1.3). Subsequently, assuming values for ON/OFF resistance state R_{on} and R_{off} the output voltage for each bit-line calculation was written in

Python. All the values defined in the crossbar structure such as number of rows, number of columns etc. are parameterized.

This Python script was then incorporated into a neural network to calculate the loss, accuracy and other metrics. TensorFlow open-source software library was used to design this hardware-neural network model. MNIST data was passed through the crossbar to validate the model. The input to the neural network (X_i) would be mapped as input voltages (V_i), while the updating weights ($W_{i,j}$) would correspond to conductance ($G_{i,j}$) of the resistive element used in the hardware. The output voltage (N) of the crossbar through each bit-line would then correspond to output neuron (N).

The mapping of Neural Network to the crossbar array is shown in Figure 1.7. $V_1, V_2, \ldots$ V_N are the input voltages and $N_1, N_2, \ldots$ N_N are the output voltages. The nodes in the crossbar correspond to the conductance values.

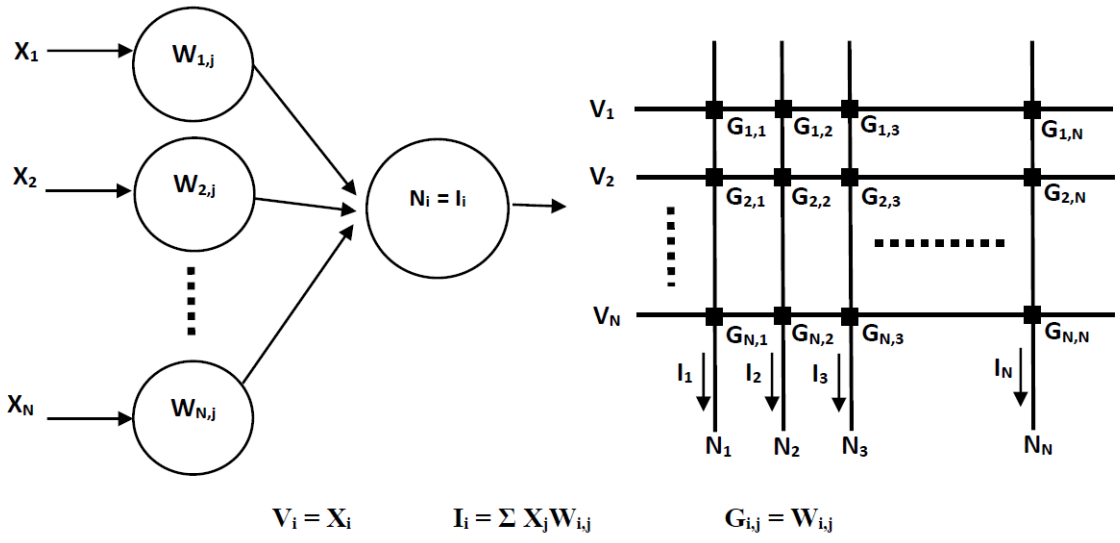


Figure 1.7: Mapping of Neural Network to Crossbar Array

In the paper by Zhang, Cosemans & Wouters et al.[29], the impact of resistive switching element (RSE) parameters is investigated for read performance of a two-terminal one-selector one-resistor cell with SPICE simulations.

Another paper by the same authors [30] investigates the impact of selector characteristics on the overall 1S1R cell performance. This research is similar to the research being conducted and discussed in this paper but differs in the way that the paper by Zhang, et al.[29]-[30] focuses on the effect of OFF/ON resistance ratio on read margin whereas this paper focuses on the hardware-neural network incorporation aspect and its effect on accuracy.

This chapter talked about the problem statement in general providing the hardware aspect of the research problem.

The next chapter describes the software aspect of the research problem including background information on CNNs, image classification, benchmarks and other resources used.

Chapter 3 describes the hardware-neural network model including detailed explanation of the flow of the model.

Chapter 4 gives a detailed report of the results obtained and the subsequent discussions.

Chapter 5 concludes the work and possible future extension of the research.

CHAPTER 2

CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural network (CNN) is a class of deep neural networks used mainly for image recognition and classification. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. The next section describes CNNs in detail with respect to image classification.

2.1 Image Classification

Image classification is a supervised learning problem where we take an input and output a class or a probability that the input belongs to a particular class by training a model to recognize them using labeled example photos. The labeled example photos are commonly known as training set and the images used to test the model are known as test set.

Figure 2.1 shows a simple image classification example.

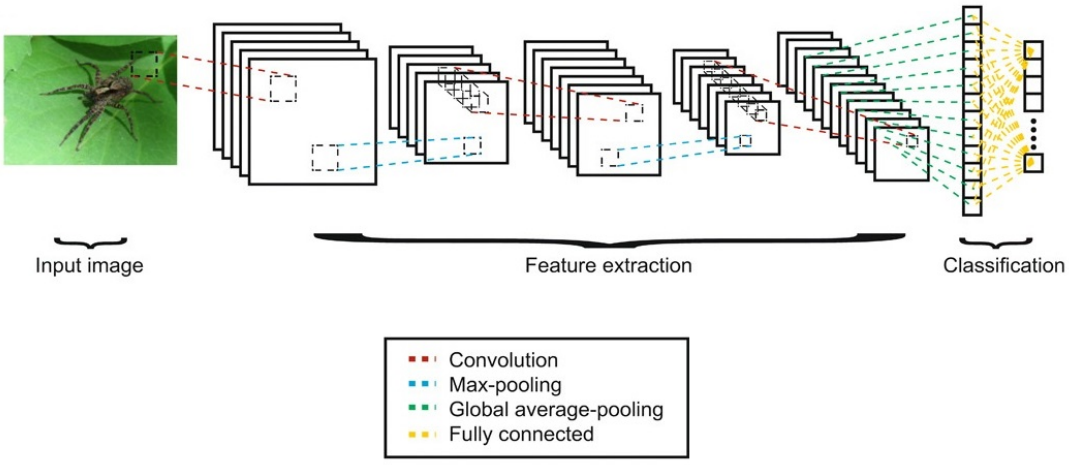


Figure 2.1: Simple Image Classification. Taken from: Nasr, et al.[31], Figure 1(A)

As we can see from Figure 2.1, CNNs can be broadly divided into feature extraction or feature learning and classification. Feature extraction consists of different layers such as convolution and pooling layers where various features in an image are extracted by the neural network model and the model is updated during the training process to predict better. Fully connected layers are a part of classification where the images are classified and appropriately labeled.

2.1.1 Convolution Layers

Convolution layers are used to extract features from images by applying filters. The vector of weights and the bias are called filters and represent particular features of the input. As the network is trained, these layers continuously keep updating their filters which are composed of small kernels. A feature map is generated for each filter which is then passed over an activation function to decide whether a certain feature is present at a given location in the image. ReLU (Rectified linear unit) is a commonly used activation function which makes the model easy to train and often achieves better performance.

2.1.2 Pooling Layers

Pooling layers are used to down-sample feature maps. They reduce the dimensionality of the network. Most common pooling methods are max pooling and average pooling. In max pooling, the largest values on the feature maps are selected and used as inputs to subsequent layers. In average pooling, average of all values on the feature maps are selected as inputs to subsequent layers. This provides an abstract form of representation and helps in-part to curb over-fitting.

2.1.3 Fully Connected Layers

Fully connected layers basically connect every neuron in one layer to every neuron in the next layer. The output of pooling layer is first flattened and then passed through fully

connected layer for classification. An activation function, typically Softmax, is used to output a vector that represents the probability distributions of a list of potential outcomes. These outcomes form the output layer where each neuron represents a classification label.

In general, the first few layers learn low-level features and detect the basic features in an image. The middle layers learn mid-level features and detect parts of objects. The last layers learn high-level features and detect full objects.

2.2 TensorFlow and Keras Integration

To develop and train the neural network model for this work, we use TensorFlow machine learning library. Keras, which is a neural network library, runs on top of TensorFlow. The following sections explain more about TensorFlow and Keras and how we integrate these two resources to use them to our advantage in this work.

2.2.1 TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Large scale neural networks can be created with many layers.

TensorFlow computations are expressed as stateful dataflow graphs. TensorFlow performs operations on multidimensional data arrays which are referred to as tensors and hence the name TensorFlow. It can be used as a general hardware acceleration library and Tensor processing unit (TPU) which is an application-specific integrated circuit was built specifically for machine learning and tailored for TensorFlow[32].

2.2.2 Keras

Keras is an open-source neural-network library written in Python. TensorFlow's high-level APIs are based on the Keras API standard for defining and training neural networks. Keras enables fast experimentation with various neural networks.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, dropout, batch normalization, pooling, etc. to simplify the neural network code for implementing image and text data classification. It is useful in quickly building complex models.

Keras also provides direct access to some datasets used in this work such as MNIST, CIFAR-10 and CIFAR-100. Keras makes it very easy to import and split the datasets into train and test sets.

2.2.3 Integration

In our hardware-neural network model, we design the model using the Keras + TensorFlow integration built directly into the TensorFlow library. The model is defined and datasets are imported using Keras and TensorFlow is used to implement specific TensorFlow functionality to custom build various layers in the network. This gives more control over the network.

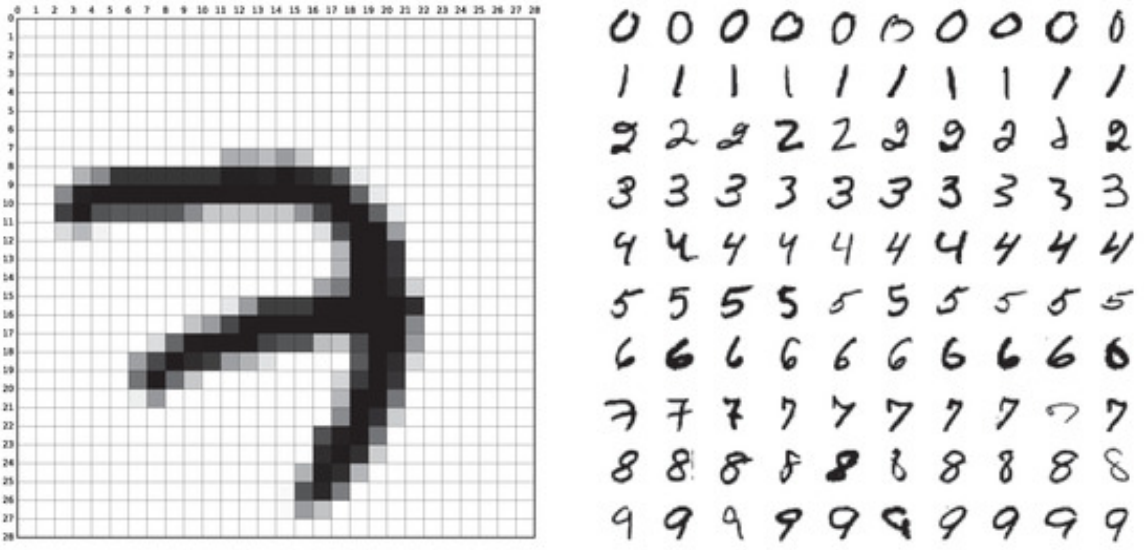
2.3 Benchmark Datasets Used

In this work, 5 different types of datasets, namely, MNIST, CIFAR-10, CIFAR-100, ImageNet and ImageNet-64, were used as benchmarks to train and test the hardware-neural network model for image classification. These benchmarks progressively increase in size and complexity in classification of images. Characteristics of each dataset are explained below.

2.3.1 MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a huge set of binary greyscale handwritten digits which is commonly used for training various image processing systems as well as in machine learning for training and testing networks[33]-[34]. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

A sample of the MNIST dataset with handwritten digits is shown in Figure 2.2.



(a) MNIST sample belonging to the digit '7'. (b) 100 samples from the MNIST training set.

Figure 2.2: Taken from: Baldominos, et al.[35], Figure 1

MNIST dataset contains 60,000 training images and 10,000 testing images. Each image is of size 28×28 pixels. The images are greyscale and hence their RGB values range from 0 to 255. These images also contain labels ranging from 0 to 9 classifying them into digits 0 to 9.

2.3.2 CIFAR-10

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is another dataset with a collection of images used for training machine learning algorithms. It is a subset of the

80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton[36].

The CIFAR-10 dataset contains 60,000 color images of size 32 $\times$ 32 belonging to 10 different classes with labels as shown in Figure 2.3.

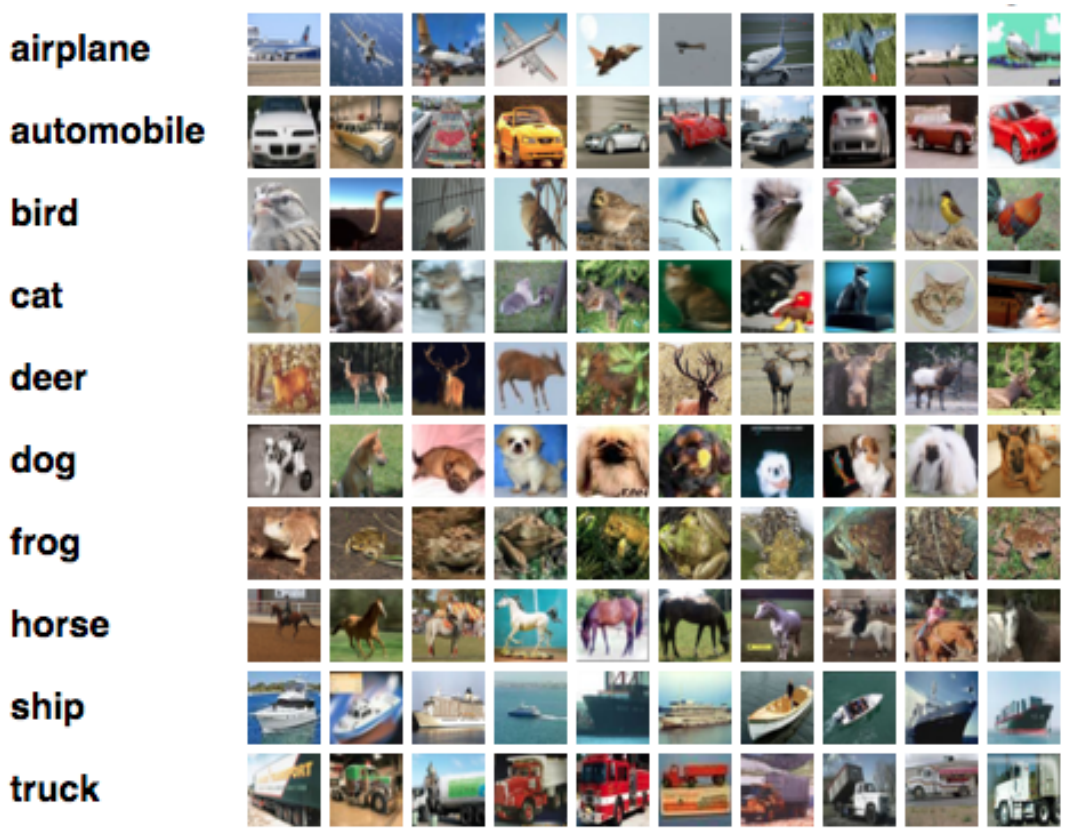


Figure 2.3: Sample of CIFAR-10 dataset. Taken from: Krizhevsky[36]

Training set consists of 50,000 images and the remaining 10,000 images are for testing. There are 6,000 images of each class and the 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

2.3.3 CIFAR-100

The CIFAR-100 dataset is similar to CIFAR-10 except that it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super-classes.

Each image comes with a ”fine” label specifying the class to which it belongs and a ”coarse” label specifying the super-class to which it belongs as shown in Figure 2.4.

Coarse labels	Fine labels
Fish	Aquarium fish, flatfish, ray, shark, trout
Flowers	Orchids, poppies, roses, sunflowers, tulips
Food containers	Bottles, bowls, cans, cups, plates
Trees	Maple, oak, palm, pine, willow
People	Baby, boy, girl, man, woman
Vehicles 1	Bicycle, bus, motorcycle, pickup truck, train
Vehicles 2	Lawn mower, rocket, streetcar, tank, tractor
Reptiles	Crocodile, dinosaur, lizard, snake, turtle

Figure 2.4: Example of CIFAR-100 Classification Labels. Taken from: Mu, et al.[37], Table 1

2.3.4 ImageNet

ImageNet is a very large database containing more than 14 million hand annotated images designed for use in visual object recognition software research.

ImageNet contains more than 20,000 categories consisting of several hundred images in each category. The images vary in dimensions and resolution. The average image resolution on ImageNet is 469×387 pixels. Generally, pre-processing is done on the images to sample them to 224×224 pixels. A sample of the ImageNet dataset is shown in Figure 2.5.

2.3.5 ImageNet-64

ImageNet-64 is similar to ImageNet dataset, except that the images are resized to 64×64 pixels for faster processing. It has the same number of total images in the dataset but the resolution of each image is lower.

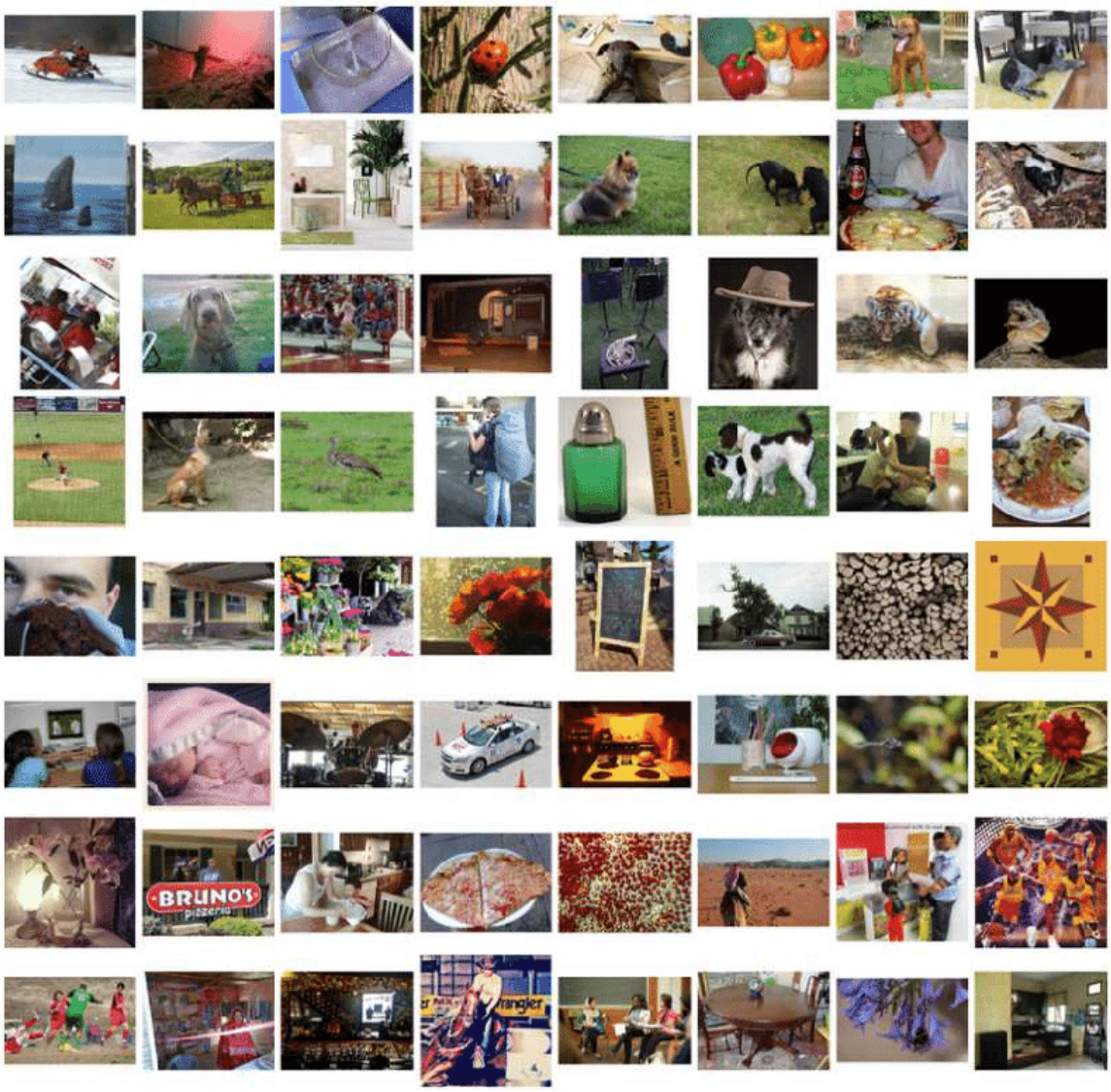


Figure 2.5: Sample of ImageNet dataset. Taken from: Chen, et al.[38], Figure 2

Since the images are down-sampled, the final accuracy of ImageNet-64 will not be as high as that of original ImageNet dataset. This dataset can be used to validate any neural network model which needs to be trained with ImageNet data. Since training the hardware model on the huge ImageNet dataset would take days, it can first be tested on ImageNet-64 dataset to correct any probable errors which might disrupt or terminate the training process.

2.3.6 Overview of the Datasets

The benchmark datasets can be summarized as shown in Table 2.1.

Table 2.1: Summary of Datasets

Dataset	Image Size	Number of Images	Number of Classes
MNIST	28×28	70000	10
CIFAR-10	32×32	60000	10
CIFAR-100	32×32	60000	100
ImageNet-64	64×64	14000000	20000
ImageNet	Variable	14000000	20000

As we move down Table 2.1, the complexity of the datasets increases, making them harder to be predicted by neural networks. Thus the accuracy is expected to decrease for the larger datasets.

CHAPTER 3

HARDWARE-NEURAL NETWORK MODEL DESCRIPTION

This chapter gives a detailed description of the hardware-neural network model defined and used in this work to train all the benchmark datasets described in the Section 2.3. The user inputs, layers and activation functions used and final output are explained in separate sections. The hardware-neural network model was trained using Nvidia GeForce RTX 2080. Flow of this whole procedure is shown in Figure 3.1.

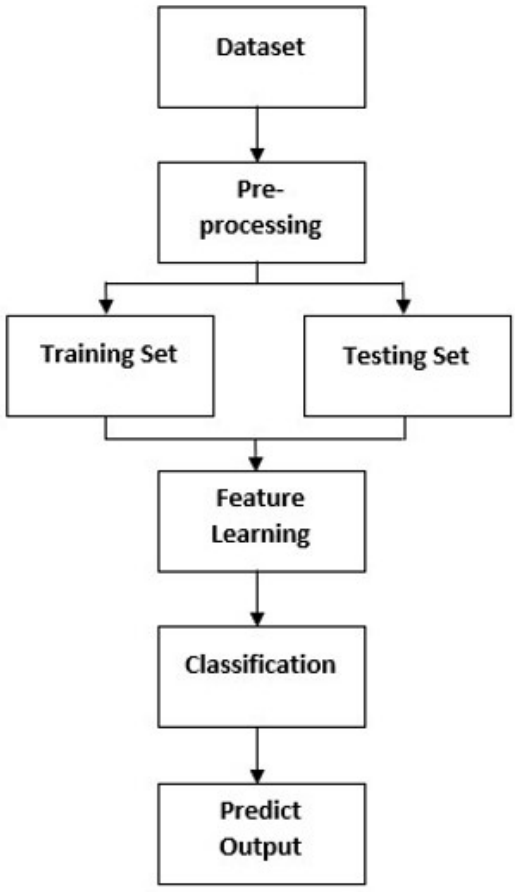


Figure 3.1: Flowchart of Hardware-Neural Network Model

3.1 Input to the Hardware-Neural Network Model

The inputs to the model are parameterized. Following inputs are taken from the user:

1. **Dataset** - Choose the dataset to train the hardware-neural network model and import it using Keras. Pre-process the data by splitting it into training set and test set. Since each dataset has different image size, number of input images and output labels, the convolution layers and fully connected layers should be sized accordingly.

2. **Epoch** - Epoch is the number of complete passes through the entire dataset. We pass the entire dataset multiple times through the neural network to get optimal fitting of the curve.

   One epoch leads to underfitting whereas too many epochs lead to overfitting. Goal is to find the optimal number of epochs for optimal fitting of the curve. This concept is explained in Figure 3.2.

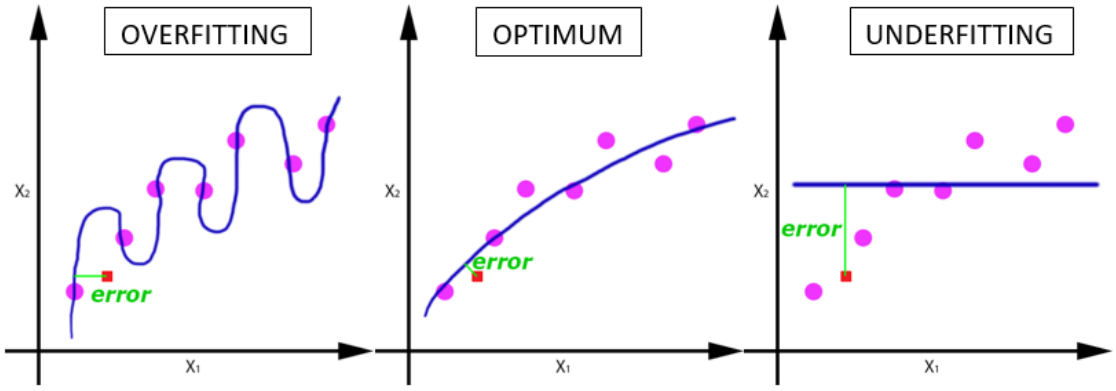


Figure 3.2: Curve Fitting. Taken from: [39]

3. **Batch size** - Batch size sets the number of samples to be trained before updating the weights in the model. Sending data in batches results in faster training process and requires less memory in case the dataset is too big to fit in memory.

4. **Goff** - Inverse of resistance value R_{off} gives us the conductance G_{off}. We prefer G_{off} instead of R_{off} since it is easier to use in calculations.

$$G_{off} = \frac{1}{R_{off}} \tag{3.1}$$

5. **Ratio** - R_{off} to R_{on} ratio is translated to G_{on} to G_{off} and this ratio is multiplied by G_{off} to get G_{on}. Instead of manually calculating G_{on} every time, it has been parameterized and providing ratio and G_{off} parameters is enough to calculate the value of G_{on}. G_{off} and G_{on} values are used to restrict the weights of the model since they represent the conductance of the crossbar structure and in an ideal situation, conductance should not vary.

$$\frac{R_{off}}{R_{on}} = \frac{G_{on}}{G_{off}} \tag{3.2}$$

$$G_{on} = \frac{G_{on}}{G_{off}} \times G_{off} \tag{3.3}$$

$$G_{on} = Ratio \times G_{off} \tag{3.4}$$

6. **Dropout rate** - Dropout rate defines how many nodes are randomly dropped out during training. This is a regularization technique patented by Google to reduce overfitting in neural networks.

7. **Epsilon** - Epsilon is a parameter defined in Adam optimizer which is used to train the model. Adam optimizer is explained in detail in Section 3.3.

   Epsilon is defined to avoid divide by zero error while updating the weights. Small epsilon results in larger weight updates whereas a big epsilon results in smaller weight

updates. Thus, it is important to select a right value of epsilon.

8. **Learning rate** - Learning rate, also known as step size, is the amount by which the weights are updated while training. Choosing a small value results in long training process whereas a large learning rate can lead to a fast and unstable training process with sub-optimal set of weights. This trend can be seen in Figure 3.3.

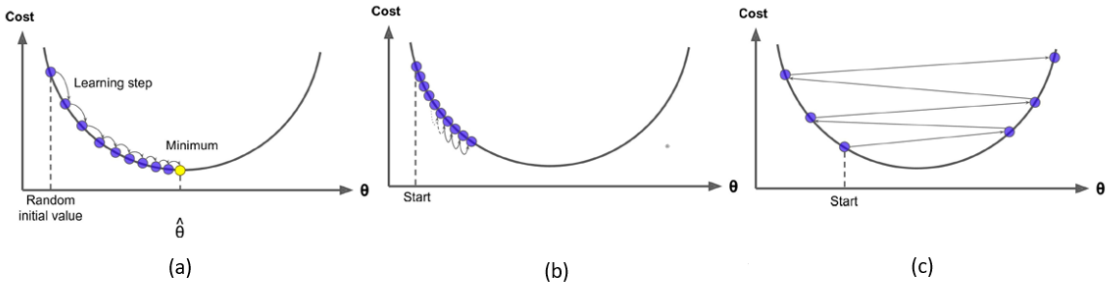


Figure 3.3: Learning Rate. Taken from: Géron[40]

Figure 3.3(a) shows the curve for optimal learning rate. Figure 3.3(b) shows a slow learning rate where the model converges to minimum cost but very slowly. Figure 3.3(c) shows a fast learning rate where the model may not converge.

9. **Standard deviation** - Standard deviation defines how wide the given distribution of data is. It shows the dispersion from mean. Low standard deviation means data points are close to the mean whereas high standard deviation means data is spread out over a range. Standard deviation is used in this hardware-neural network model to add uncertainty. More explanation about the usage and computation of standard deviation is provided in Section 3.4.

3.2 Convolution Layers Used

Feature extraction and classification methods used in the hardware-neural network model are explained below.

3.2.1 Feature Extraction

In this hardware-neural network model, feature extraction is divided into blocks. Each block consists of 2 convolution layers which are batch normalized and then passed through a pooling layer.

As explained in Section 2.1.1, convolution layers consist of filters which generate feature maps. Two types of filters are used - one for positive values and one for negative values. This is because standard deviation parameter in the definition of filter leads to negative values. Since we cannot model negative current in actual hardware, we split them into two types of filters and take their difference.

This convolution layer is then batch normalized. Batch normalization is used to normalize the input layer by adjusting and scaling the activations. Batch normalization allows each layer of a network to learn by itself a little bit more independently than other layers. It also reduces overfitting.

The normalized layer is passed through ReLU activation function. It is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. Negative values will become zero and not be mapped appropriately. Hence it is very important to split the filters and make sure there are no negative values which in turn could decrease the ability of the hardware-neural network model to train from the data. ReLU function is defined as:

$$f(x) = max(0, x) \tag{3.5}$$

where x corresponds to the features in an image. Graphical plot of this function is shown in Figure 3.4.

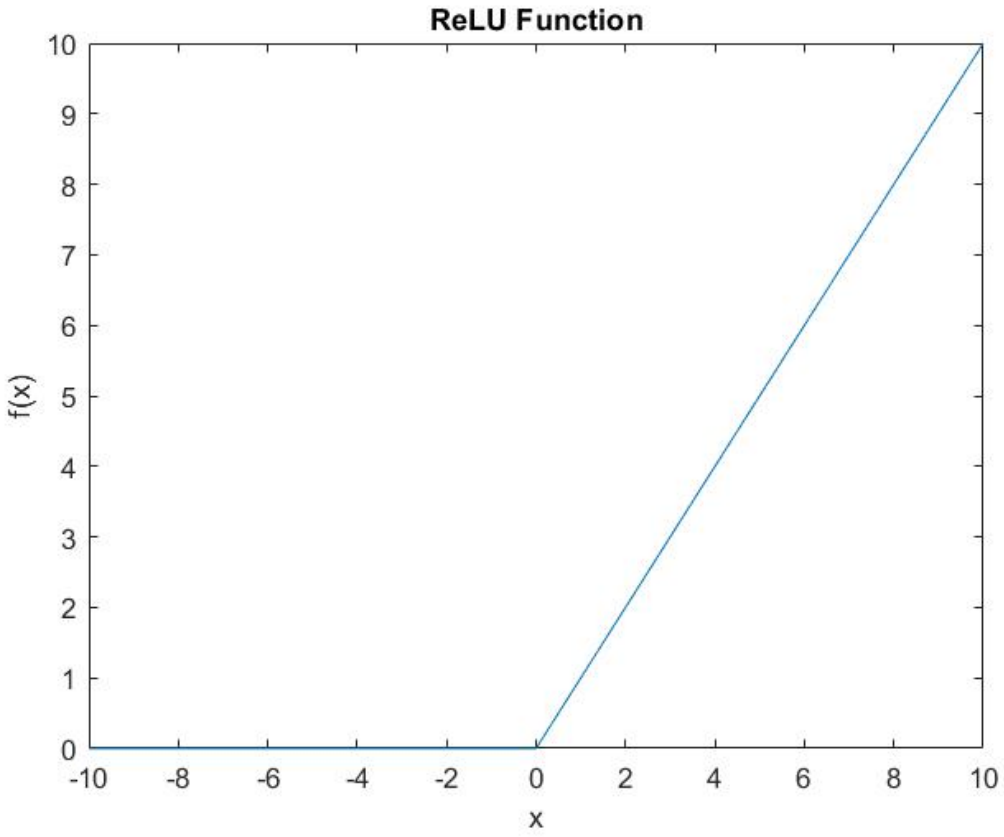


Figure 3.4: MATLAB plot of ReLU Activation Function

The final layer in the block is a pooling layer. Average pool is used to down-sample and compute the average value of each region. This reduces the number of connections to the following layers.

5 blocks are used for benchmarks MNIST, CIFAR-10, CIFAR-100 and ImageNet-64. Since each block consists of 2 convolution layers, these models contain 10 convolution layers and a fully connected layer.

For ImageNet dataset, 14 blocks are used. Thus it contains 28 convolution layers and a fully connected layer. The number of blocks is based on the complexity of benchmark dataset and are selected to give optimal train and test accuracy values.

Accuracy improves with wider networks as shown by Zagoruyko, et al.[41] in Figure 3.5. This further reinforces our proposal of using 29 layers for training the hardware-neural network model using ImageNet dataset.

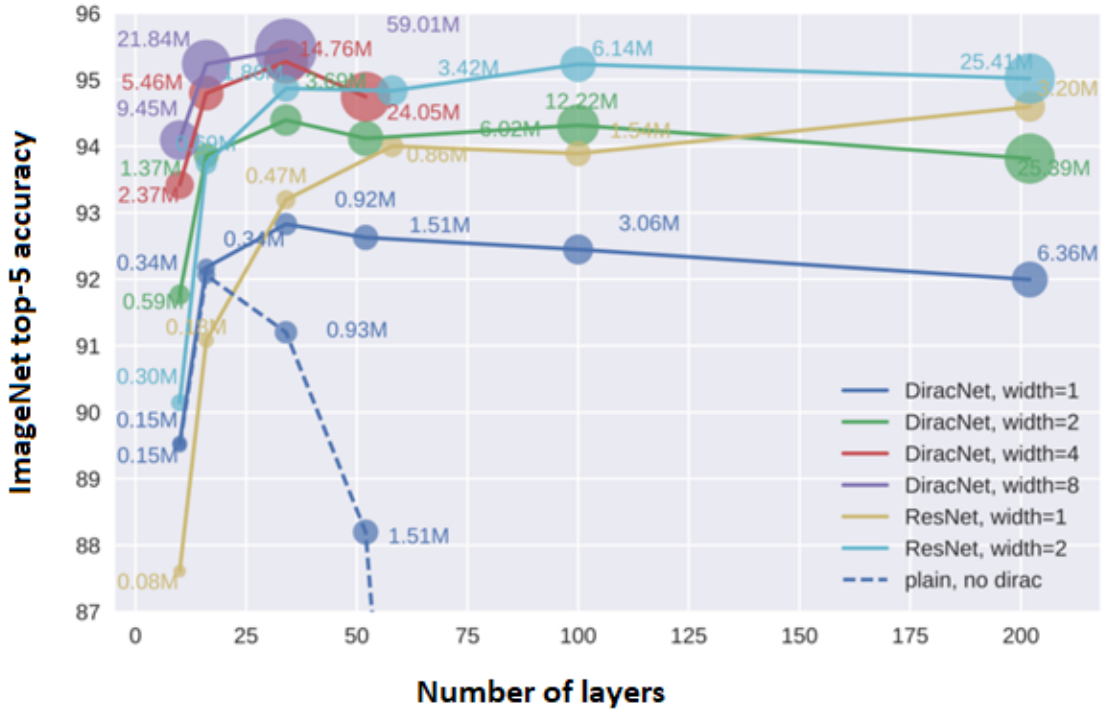


Figure 3.5: ImageNet Accuracy vs Number of Layers. Taken from: Zagoruyko et al.[41], Figure 1. Courtesy of Jae-sun Seo

ResNet[42] refers to residual neural network which are deep neural networks with some shortcut connections. These connections skip some layers and the model ends up successfully training deep neural networks.

DiracNet[41] is a deep neural network based on ResNet but without explicit shortcut connections. This architecture uses a novel weight parameterization called Dirac parameterization to achieve an efficient deep neural network.

3.2.2 Classification

As explained in Section 2.1.3, the output of the pooling layer is reshaped and flattened in this stage. This is then passed through a dense fully connected layer which provides the final classification labels in the form of an output vector using Softmax function.

Softmax function takes a vector of K real numbers as input, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. The mathematical equation is given as:

$$\sigma(x)_i = \frac{\exp x_i}{\sum_{i=1}^{K} \exp x_i} \tag{3.6}$$

Graphical plot of this function is shown in Figure 3.6.

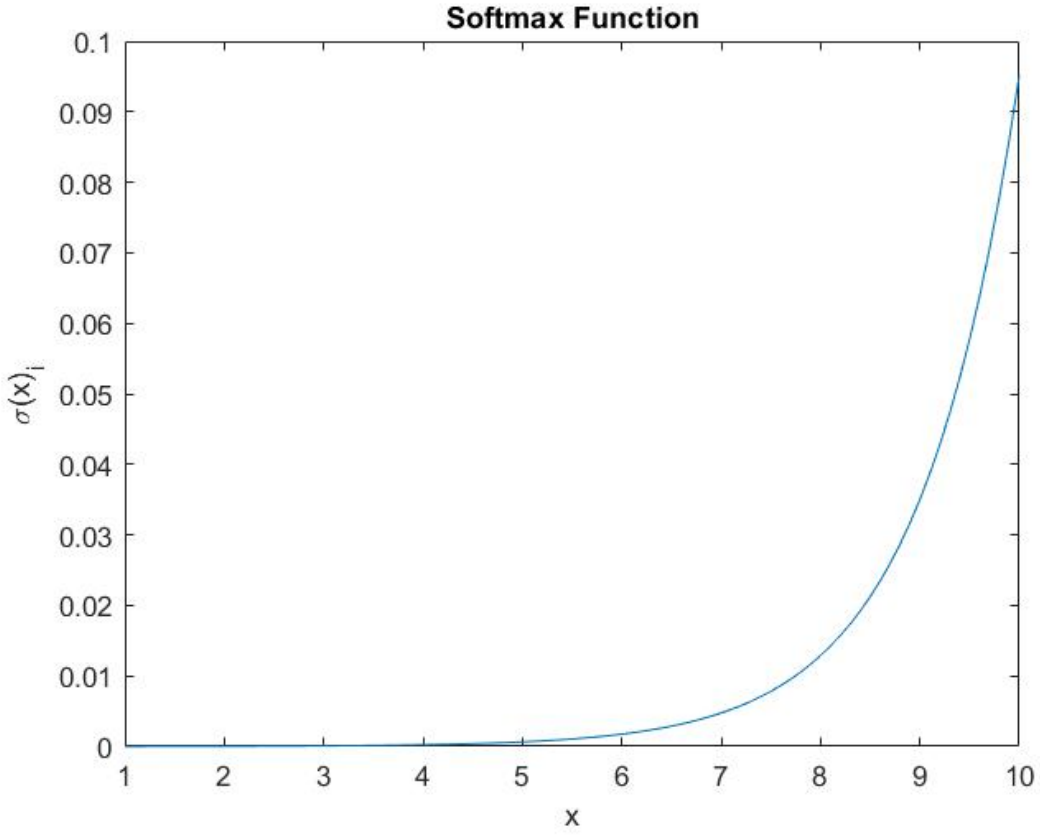


Figure 3.6: MATLAB plot of Softmax Function

The standard exponential function is applied to each element x_i of the input vector x and these values are normalized by dividing them by the sum of all these exponentials. This normalization ensures that the sum of the components of the output vector $\sigma(x)$ is 1.

3.3 Accuracy Calculation and Interpretation

The model is trained with an optimizer and loss function. The weights are modified using the optimizer function and loss function is used to better the results of optimizer by steering it in the right direction. Loss function calculates the difference between output calculated for a given input and the target variable. Optimizer uses these results to modify the weights and bias to reduce this loss value.

Adam optimizer is an optimization algorithm used in this work to iteratively update network weights during training. It uses adaptive moment estimation to converge without much fine tuning of other parameters. Adam optimizer is selected for this work because it works efficiently for large models and datasets.

In a TensorFlow session, train and test dataset are fed in batches to the hardware-neural network model through feed-dict and the accuracy values for train and test set are computed separately.

Accuracy is calculated for each batch using the equation:

$$Accuracy = \frac{Correct\,predictions\,in\,the\,batch}{Total\,predictions\,in\,the\,batch} \tag{3.7}$$

Accuracy values range from 0.0 to 1.0. Higher the accuracy, better the network in predicting labels of the test images in the given dataset.

3.4 Adding Uncertainty to the Hardware-Neural Network Model

A 10% uncertainty is added to the weights of the hardware neural network model. This is to make the model more realistic since the weights correspond to the conductance values

of the crossbar. In real world, the resistors do not always give the exact resistance we want. Thus, the conductance value changes accordingly. This uncertainty in the model is expected to reduce the accuracy values compared to a perfect model. These results are compared with the results obtained in ideal case in the next chapter.

In an ideal scenario, the distribution of data is similar to that of uniform distribution. As we can see from Figure 3.7, data is either the highest point on the y-axis amplitude, in this case it will be G_{on} , or lowest amplitude G_{off}.

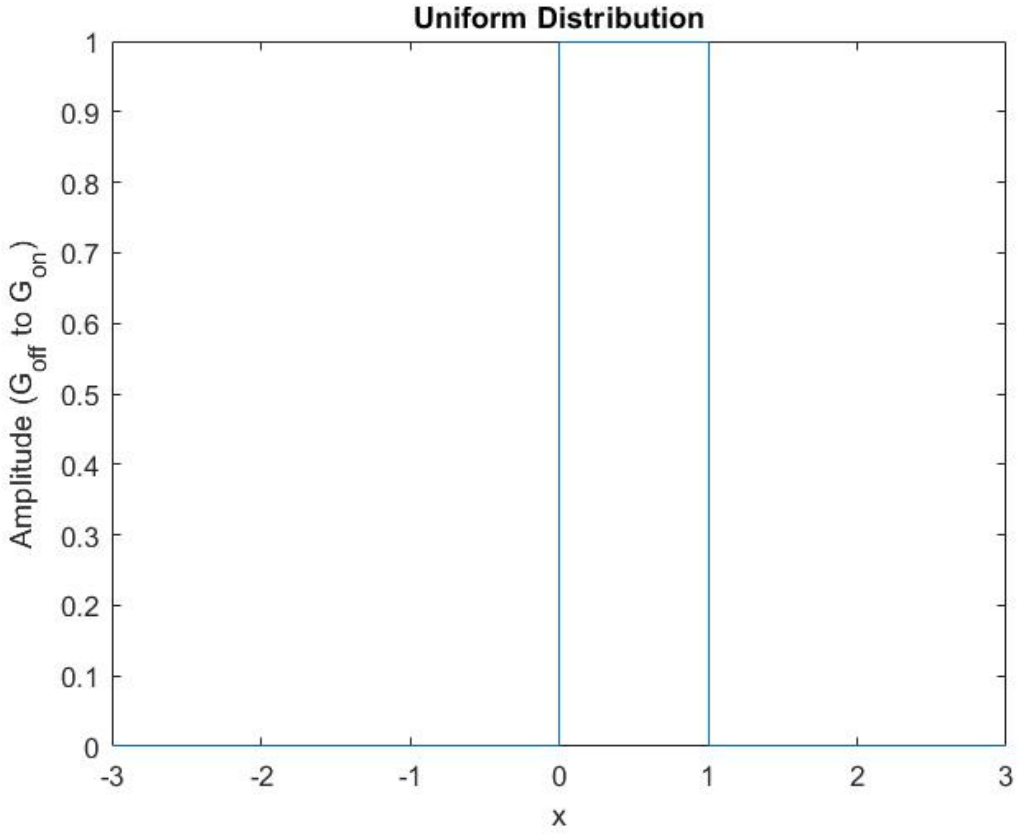


Figure 3.7: MATLAB plot of Uniform Distribution

In a non-ideal scenario, data will be distributed in a random normal manner as shown in Figure 3.8. This plot has standard deviation 1.

The amplitude of data will not always be G_{on} and G_{off} and it follows more of a normal distribution, also known as Gaussian distribution. Data points will be spread around the mean (here mean = 0) and the dispersion is defined by standard deviation. Standard

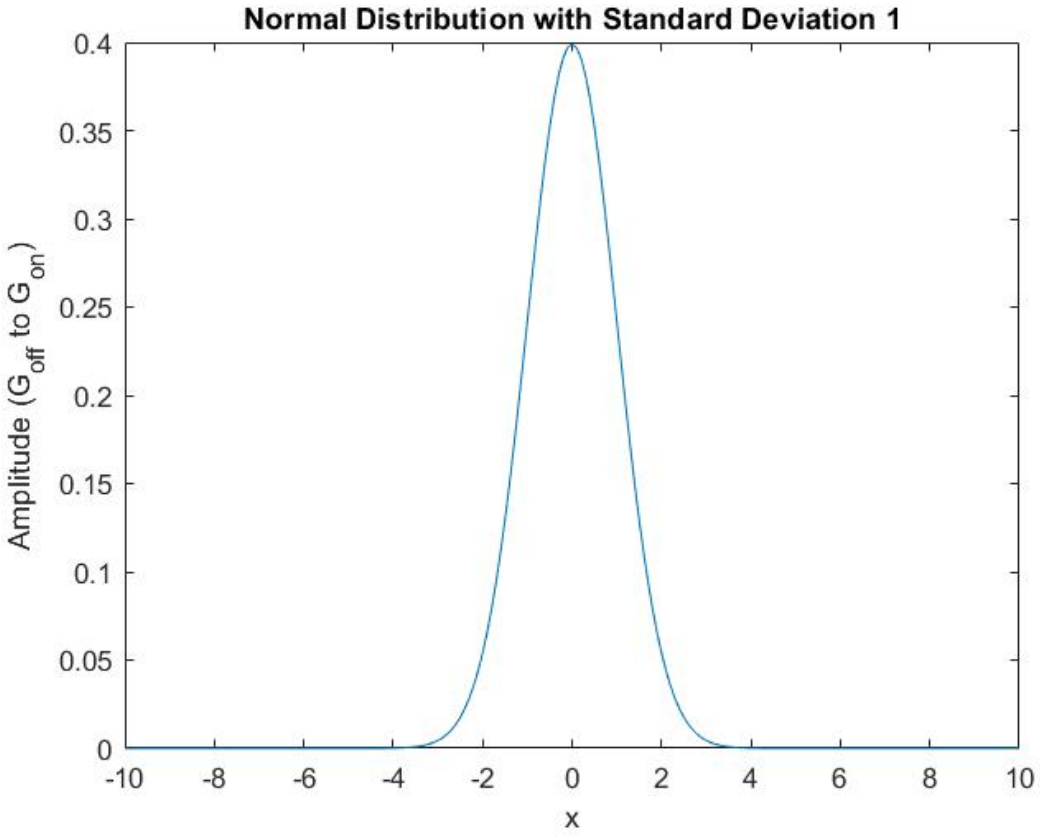


Figure 3.8: MATLAB plot of Normal Distribution

deviation defines the percentage of uncertainty in the model.

For 10% uncertainty, variance (σ^2) is defined as:

$$\sigma^2 = \frac{10}{100} = 0.1 \tag{3.8}$$

$$Standard\,deviation = \sigma = \sqrt{0.1} \tag{3.9}$$

The plot for 10% uncertainty is shown in Figure 3.9.

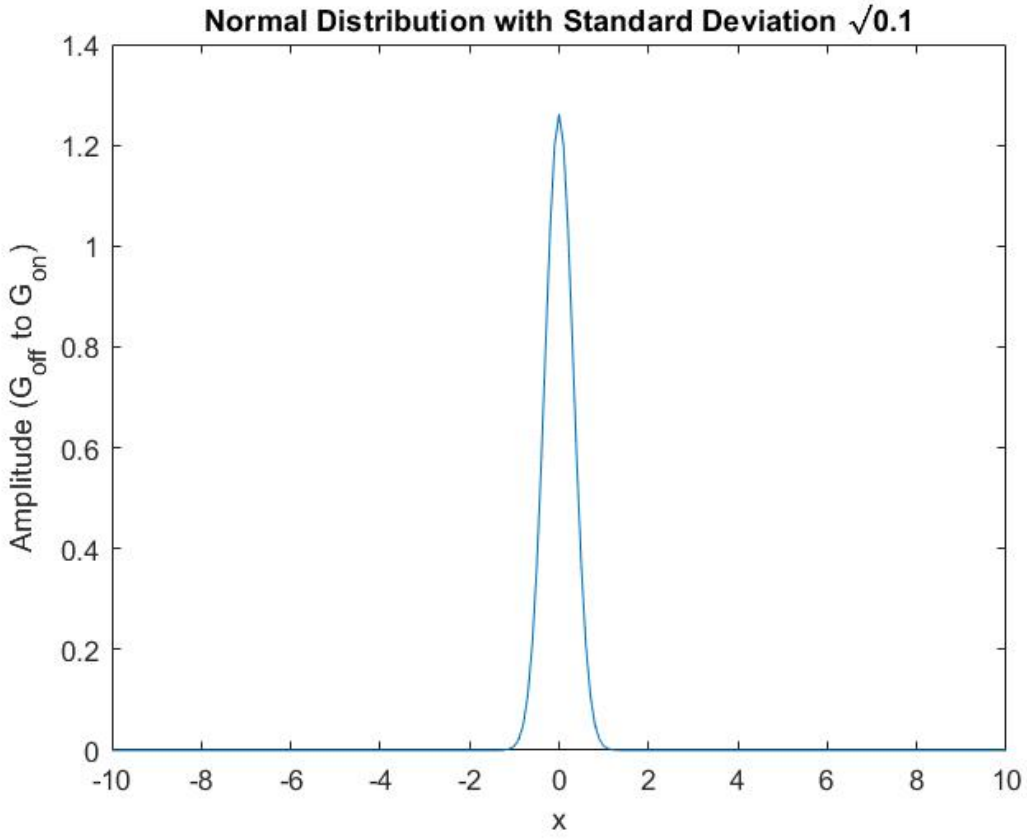


Figure 3.9: MATLAB plot of Normal Distribution with Standard Deviation $\sqrt{0.1}$

The difference between Figure 3.8 and 3.9 is that the curve is more narrow for the 10% uncertainty model and is less dispersed. The data points lie about $\pm$ 10% of G_{on} in Figure 3.9. This translates to the possible change in conductance values in a non-ideal scenario.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter focuses on the results obtained and the assumptions made in the noise analysis, ideal and non-ideal hardware-neural network model along with discussions about the results.

4.1 Noise Analysis of Crossbar Structure

As mentioned in Chapter 1.3, noise analysis was performed on the crossbar structure shown in Figure 1.6 by simulating the circuit on Cadence Spectre Circuit Simulator.

This analysis was performed for different sizes of the crossbar structure as well as for various OFF/ON resistance ratios. Noise was measured as Power Spectral Density with unit $V/\sqrt{Hz}$.

The results obtained by simulating the crossbar structure with noise analysis are shown in Table 4.1. The following values were assumed during simulation:

$R_{on} = 150k\Omega$

$R_{ref} = 20k\Omega$

The value of R_{off} changes based on the ratio as: $R_{off} = R_{on} \times Ratio$.

Table 4.1: Results of Noise Analysis

Size of Crossbar NxN	**Power Spectral Density** ($V/\sqrt{Hz}$)		
	Ratio = 4	**Ratio = 300**	**Ratio = 20000**
2	29.82×10^{-9}	1.585×10^{-6}	28.42×10^{-6}
4	39.41×10^{-9}	1.963×10^{-6}	37.32×10^{-6}
8	48.47×10^{-9}	2.269×10^{-6}	45.41×10^{-6}
16	56.95×10^{-9}	2.483×10^{-6}	52.45×10^{-6}
32	65.66×10^{-9}	2.656×10^{-6}	58.72×10^{-6}
64	75.87×10^{-9}	2.796×10^{-6}	64.48×10^{-6}
128	89.98×10^{-9}	2.92×10^{-6}	70.3×10^{-6}

From Table 4.1, we can see that the noise power spectral density increases with the increase in ratio from $nV/\sqrt{Hz}$ to $\mu V/\sqrt{Hz}$. This is expected because noise caused by current noise decreases with resistance value. As the OFF/ON resistance ratio increases, R_{off} increases and leads to more current noise.

Following the same principle, with increase in size of the crossbar, noise should increase too. From the results, we can see that the value does increase, but the increase is not too high. The probable reason for this is that since the resistors are connected in series, the same current flows through them. Hence any change in number of resistors connected does not affect current.

4.2 Ideal Hardware-Neural Network Model

Typical run-times for all the benchmark datasets used in this work are shown in Table 4.2.

Table 4.2: Run-time for 5 datasets

Dataset	**Run-time**
MNIST	6 minutes
CIFAR-10	8 minutes
CIFAR-100	10 minutes
ImageNet-64	6 hours
ImageNet	2 days 16 hours

As the dataset increases in size, the time taken to train the model increases exponentially. The reason for this is that the number of nodes in the neural network increases. In the final fully connected layer, the number of connections increases exponentially for a huge dataset like ImageNet and hence the significant increase in computation time.

The train and test accuracy values for each of the dataset for OFF/ON resistance ratios varying from 10 to 1000 are shown in Tables 4.3 - 4.7. The value of R_{off} is set to $10k\Omega$ since it was found by experimentation to give optimal accuracy values.

Table 4.3: Train and test accuracy for various OFF/ON resistance ratio for MNIST dataset

$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**	**Test Accuracy**
10k	1000	10	0.115754	0.147129
10k	100	100	0.99425	0.994753
10k	50	200	0.989671	0.993070
10k	40	250	0.989631	0.993367
10k	20	500	0.980701	0.989307
10k	13.33	750	0.967992	0.986535
10k	10	1000	0.959076	0.983070

Table 4.4: Train and test accuracy for various OFF/ON resistance ratio for CIFAR-10 dataset

$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**	**Test Accuracy**
10k	1000	10	0.090127	0.118911
10k	100	100	0.949887	0.816832
10k	50	200	0.965761	0.858911
10k	40	250	0.968216	0.830000
10k	20	500	0.978064	0.804159
10k	13.33	750	0.988171	0.797129
10k	10	1000	0.994590	0.783367

Table 4.5: Train and test accuracy for various OFF/ON resistance ratio for CIFAR-100 dataset

$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**	**Test Accuracy**
10k	1000	10	0.151952	0.229901
10k	100	100	0.841762	0.549901
10k	50	200	0.912243	0.529802
10k	40	250	0.930103	0.571683
10k	20	500	0.917215	0.568812
10k	13.33	750	0.859376	0.566337
10k	10	1000	0.806791	0.554555

Table 4.6: Train and test accuracy for various OFF/ON resistance ratio for ImageNet-64 dataset

$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**	**Test Accuracy**
10k	1000	10	0.43059	0.406213
10k	100	100	0.627413	0.446040
10k	50	200	0.627573	0.448522
10k	40	250	0.626913	0.441610
10k	20	500	0.601953	0.462380
10k	13.33	750	0.572685	0.43530
10k	10	1000	0.545314	0.427987

Table 4.7: Train and test accuracy for various OFF/ON resistance ratio for ImageNet dataset

$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**	**Test Accuracy**
10k	1000	10	0.000936	0.000918
10k	100	100	0.585263	0.544891
10k	50	200	0.610218	0.568438
10k	40	250	0.612242	0.563136
10k	20	500	0.598846	0.555322
10k	13.33	750	0.568699	0.534110
10k	10	1000	0.534575	0.492563

Figure 4.1 and Figure 4.2 show the summarized trends of these results.

From Tables 4.3 - 4.7 and Figures 4.1 - 4.2, we observe the following common trends for all the datasets:

1. For OFF/ON resistance ratio of 10, the train and test accuracy values for all datasets are very low. A low OFF/ON resistance ratio leads to bad accuracy and this is supported by theoretical evidence. Low ratio implies that R_{on} and R_{off} values are close to each other. This will make it hard for a neural network to differentiate between R_{on} and R_{off}, i.e. between binary 1 and binary 0.

2. As the OFF/ON resistance ratio keeps increasing, train and test accuracy keeps increasing till a point after which they start to decrease. The possible reason for this

observation is that high R_{off} leads to noise in the crossbar structure thus decreasing train and test accuracy.

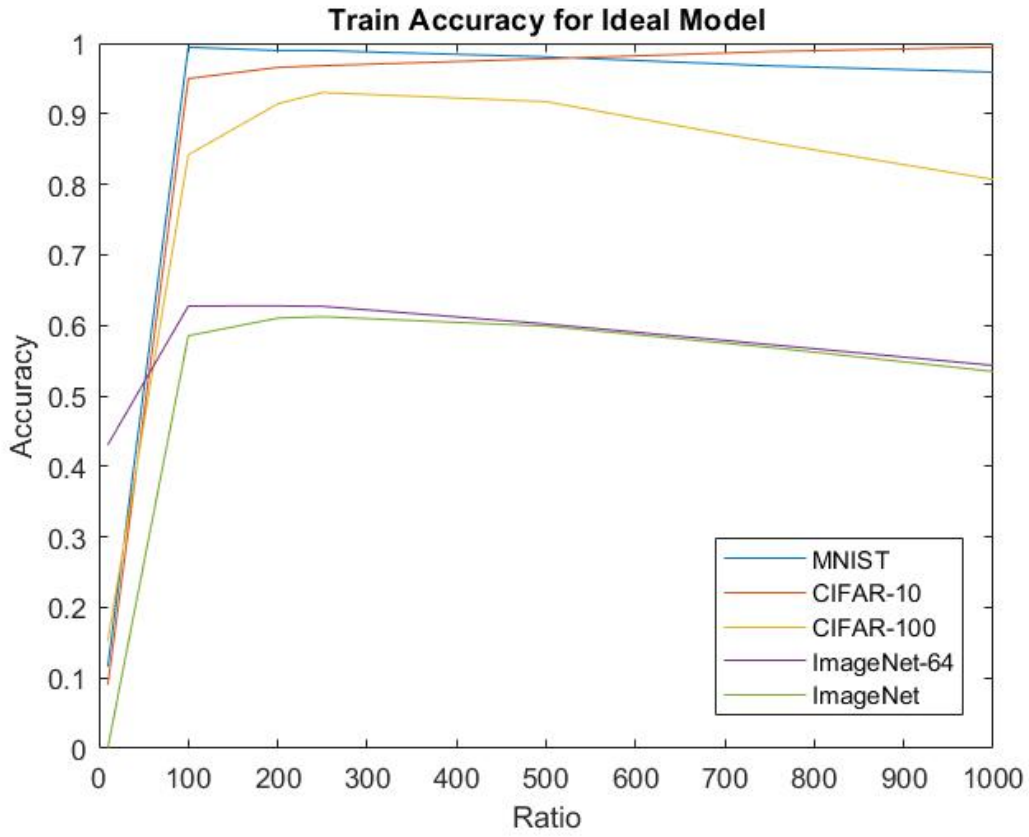


Figure 4.1: Train Accuracy for Ideal Hardware-Neural Network Model

3. The optimal OFF/ON resistance ratio for all the datasets lies around 200-500. This is the region after which the train and test accuracy values start to decrease. This range varies based on the hardware-neural network model and for this model, 200-500 is the optimal region for OFF/ON resistance ratio.

4. As the size of dataset increases, the train and test accuracy values start to decrease. This implies that the hardware-neural network model easily predicts for simple datasets MNIST and CIFAR-10. CIFAR-100 and ImageNet are more complex datasets and thus little harder to classify leading to lower accuracy.

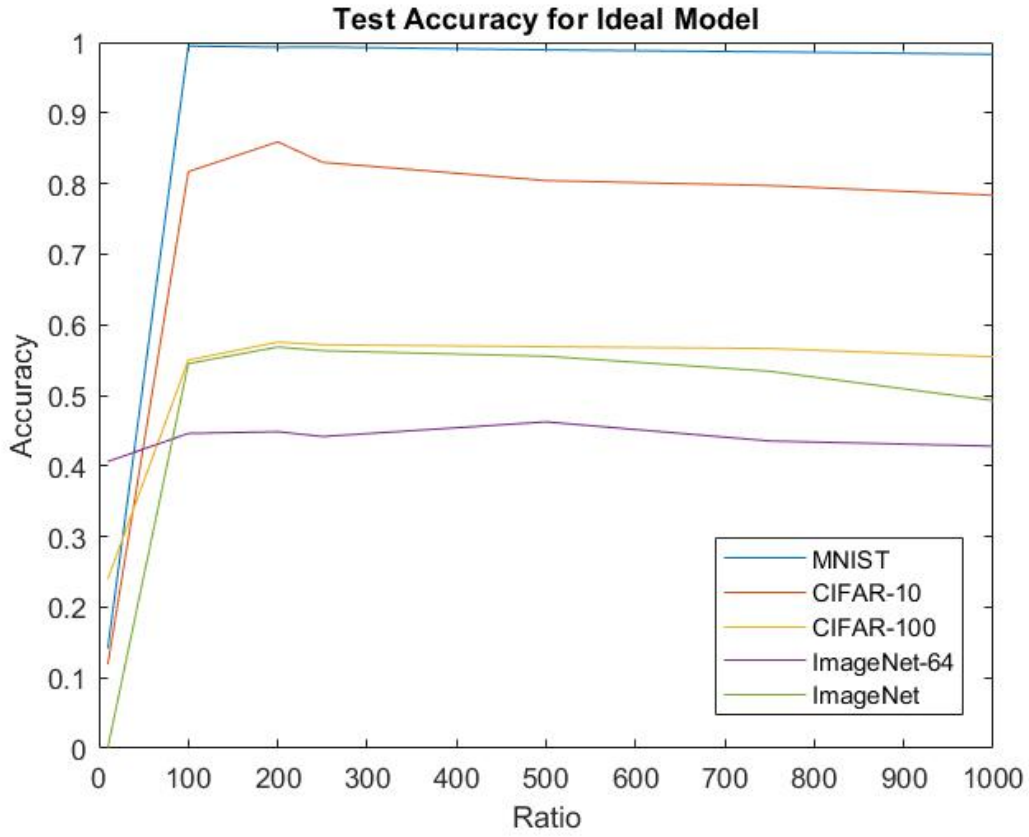


Figure 4.2: Test Accuracy for Ideal Hardware-Neural Network Model

The highest accuracy achieved till date on ImageNet dataset is 0.875 according to the paper by Xie et al.[43]. The highest accuracy achieved by this hardware-neural network model on ImageNet dataset is 0.625.

5. As noted in Section 2.3.5, test accuracy values of ImageNet-64 are lower than that of ImageNet due to down-sampled images in the dataset.

6. It is interesting to note that ImageNet gives almost same test accuracy as CIFAR-100.

In general, we observe that for the designed ideal hardware-neural network model, train and test accuracy values are reasonable. The hardware-neural network model can be better designed to get higher accuracy values for ImageNet and CIFAR-100 datasets.

4.3 Non-Ideal Hardware-Neural Network Model

The hardware-neural network model was modified and trained again for non-ideal case assuming 10% uncertainty as explained in Section 3.4.

The train and test accuracy values obtained for all the datasets are summarized and shown in Figure 4.3 and Figure 4.4. The best train and test accuracy values are tabulated in Table 4.8 and Table 4.9 respectively.

Table 4.8: Best train accuracy for all datasets

Dataset	$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Train Accuracy**
MNIST	10k	100	100	0.994349
CIFAR-10	10k	10	1000	0.994232
CIFAR-100	10k	40	250	0.950437
ImageNet-64	10k	100	100	0.686632
ImageNet	10k	50	200	0.626978

Table 4.9: Best test accuracy for all datasets

Dataset	$R_{off}(\Omega)$	$R_{on}(\Omega)$	**Ratio**	**Test Accuracy**
MNIST	10k	100	100	0.994258
CIFAR-10	10k	100	100	0.881386
CIFAR-100	10k	40	250	0.576334
ImageNet-64	10k	50	200	0.469684
ImageNet	10k	40	250	0.578285

We make the following observations from Figures 4.3 - 4.4:

1. Similar to ideal hardware-neural network model, the train and test accuracy values for OFF/ON resistance ratio of 10 are low due to difficulty in differentiating between binary 1 and binary 0.

2. Train accuracy values increase till a point in OFF/ON resistance ratio and then decrease as the ratio increases. This is also similar to the trend observed in the ideal

hardware-neural network model.

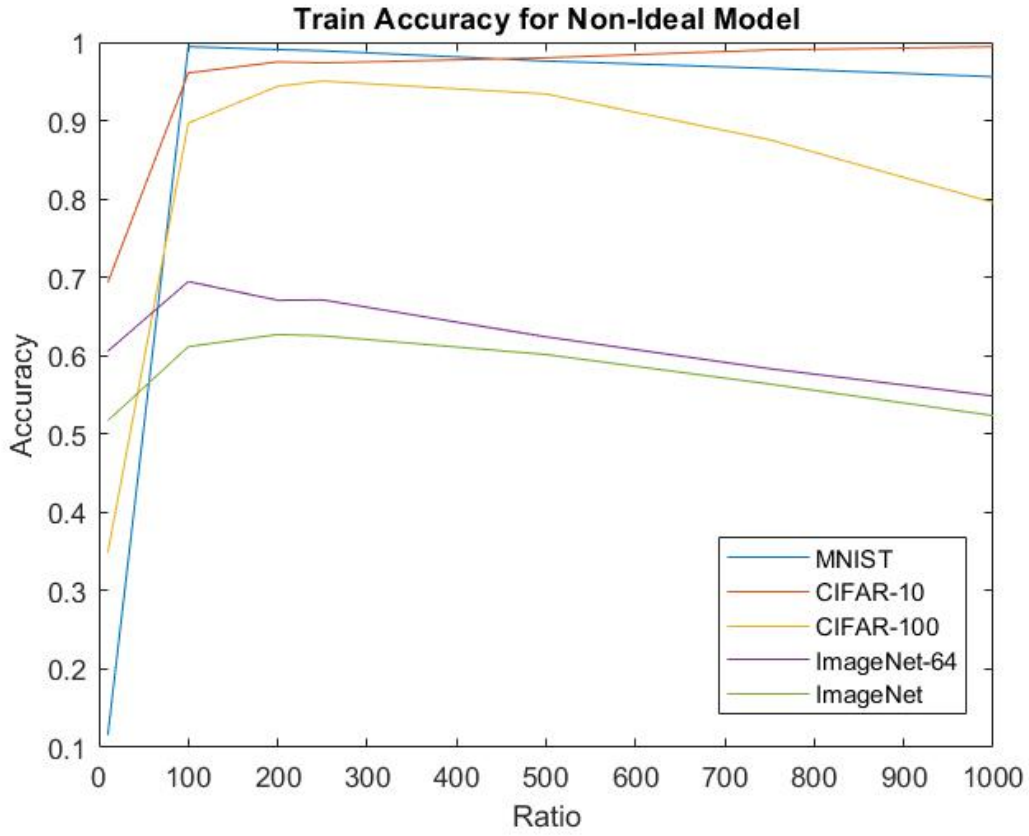


Figure 4.3: Train Accuracy for Non-Ideal Hardware-Neural Network Model

3. Test accuracy values follow the same pattern as ideal hardware-neural network model for MNIST, CIFAR-10 and ImageNet-64. The test accuracy trend for CIFAR-100 and ImageNet

   Thus the neural network model predicts some images correctly and some incorrectly. This affects the test accuracy leading to the erratic results.

4. As observed in the ideal neural network model, increase in size of dataset decreases the train and test accuracy values due to increasing complexity of the benchmark datasets. This trend can be clearly observed in Tables 4.8 - 4.9.

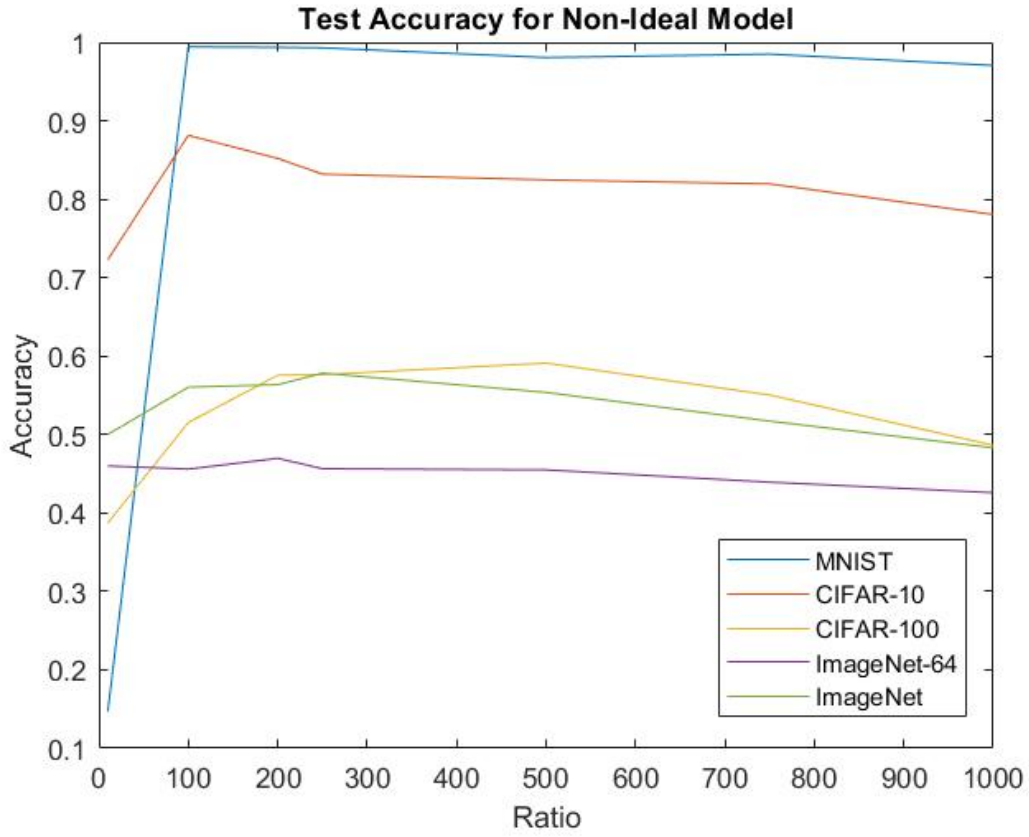


Figure 4.4: Test Accuracy for Non-Ideal Hardware-Neural Network Model

5. ImageNet gives similar results as CIFAR-100. ImageNet gives higher test accuracy value than CIFAR-100 for OFF/ON resistance ratio 250.

The major change we observe in non-ideal hardware-neural network model as compared to the ideal one is that the test accuracy values are variable. Apart from that, the hardware-neural network model gives reasonable results even for non-ideal situations.

CHAPTER 5
CONCLUSION AND FUTURE RESEARCH

This chapter summarizes the whole work and presents the conclusions drawn from the results discussed in Chapter 4. Possible future extension of this work is also discussed.

5.1 Conclusion

This research proposed a design of hardware-neural network model for a RRAM-based crossbar structure. The model was analyzed for ideal and non-ideal conditions of varying OFF/ON resistance ratios and the model was validated on benchmarks. It can be concluded that OFF/ON resistance ratios in a particular range give good results for the proposed hardware-neural network model.

The proposed hardware-neural network model addresses the memory wall problem by incorporating binary RRAM-based crossbar structure while processing huge datasets. Basic working of a neural network based on McCulloch-Pitts model was explained. The weights get updated according to given input to predict the correct output. The functionality of mapping resistances on-to weights of a neural network and mapping voltages on-to inputs to obtain current as output has been mathematically verified using MATLAB as well as by running simulations of crossbar structure designed using Cadence Virtuoso on Cadence Spectre Circuit Simulator.

Image classification involves assigning a label to an image to classify it into a predefined class. This is done using deep neural networks which are trained to extract features and then classify the images into their respective classes. Various layers in the feature extraction and classification stages of a neural network were explained. TensorFlow was used to define the neural network used in this work. Benchmark datasets were used to train the model. Keras was used to import these benchmark datasets.

The hardware-neural network model designed in this research was explained in detail along with the flow of the whole process. Various input parameters had to be properly selected for optimal fitting of the curve. Four of the benchmarks were 11-layer neural networks and the huge ImageNet was a 29-layer neural network model. The number of layers were selected to get the best train and test accuracy values. Activation functions are used to make the inputs capable of learning and predicting better by introducing non-linearity in the neurons. Optimizer function trains the model by continuously modifying weights to better predict the output.

The proposed neural network model also considered a non-ideal scenario where there is uncertainty in the conductance values of the resistors and hence uncertainty in the weights of the hardware-neural network model. This was done by modifying the data distribution from uniform to normal distribution with the amount of dispersion defined by standard deviation.

The neural network model was designed for ideal scenario where the weights and conductances are always exactly as described and this model was tested for non-ideal conditions. If the model was designed to be optimized for non-ideal scenario instead, the train and test accuracy values could have been higher.

Results were obtained by performing noise analysis on a crossbar structure using Cadence Spectre Circuit Simulator. The noise power spectral density values were found to increase with increase in size of the crossbar as well with increase in OFF/ON resistance ratio.

Run-time of training and testing the hardware-neural network model for each dataset increased with increase in size of dataset whereas the accuracy values decreased with increase in size of dataset. This was found to be valid for both ideal and non-ideal hardware-neural network models.

The hardware-neural network model presented in this work is not the best possible model and can be better designed. The possible point of break-down of this hardware-

neural network model along with future extension of this research is presented in the next section.

5.2 Future Research

The proposed model will fail if the OFF/ON resistance ratio is too small. Care must be taken to make sure there is significant difference between low-resistance-state and high-resistance-state for proper recognition of binary 1 and binary 0 by the hardware-neural network model.

As future extension, the hardware-neural network model can be modified to give better train and test accuracy values for ImageNet dataset or a more complex dataset. Modifications can be made assuming the more complicated dataset contains blurred or out-of-focus images which are harder to classify.

Another extension can be to re-design the hardware-neural network model to get high train and test accuracy for normal distributed weights with variable uncertainty. This will model the design to be more realistic where the weights or conductances can be variable and this in turn affects the accuracy.

The proposed hardware-neural network model can be implemented on actual hardware after further research and results such as power consumed and current flowing through the circuit can be obtained. The model can be further extended to work as a low-power design using the results obtained in this work since the power consumed increases linearly with accuracy for high OFF/ON ratio. This is due to the increase in leakage current with R_{off}.

On-going research on reducing sneak path current can be combined with this model to make sure there is reliable recognition between binary 1 and binary 0. Sneak path current is the current flowing through unintended paths and is a very common problem faced in crossbar structures. Sneak path current also leads to power consumption.

REFERENCES

[1] Wm. A. Wulf and Sally A. McKee, "Hitting the memory wall: implications of the obvious", *ACM SIGARCH Computer Architecture News*, Volume 23 Issue 1, March 1995, Pages 20 - 24. DOI: http://dx.doi.org/10.1145/216585.216588

[2] David A. Patterson, Andrea C. Arpaci-Dusseau, *Computer Architecture: A Quantitative Approach by John L. Hennessy*, Fourth edition, 2007.

[3] An Chen, "A review of emerging non-volatile memory (NVM) technologies and applications", *Solid-State Electronics*, Volume 125, November 2016, Pages 25-38. DOI: https://doi.org/10.1016/j.sse.2016.07.006

[4] Shubham Jain, Abhronil Sengupta, Kaushik Roy and Anand Raghunathan, "RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars". https://arxiv.org/abs/1809.00072

[5] S. Liu, Y. Wang, M. Fardad and P. K. Varshney, "A Memristor-Based Optimization Framework for Artificial Intelligence Applications", *IEEE Circuits and Systems Magazine*, Volume 18, Issue 1, 2018, Pages 29-44. DOI: https://doi.org/10.1109/MCAS.2017.2785421

[6] Geoffrey W. Burr, Robert M. Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, Lucas L. Sanches, IremBoybat, Manuel Le Gallo, Kibong Moon, Jiyoo Woo, Hyunsang Hwang & Yusuf Leblebici, "Neuromorphic computing using non-volatile memory", *Advances in Physics*, vol. X, 2:1, 2017, pp. 89-124. DOI: https://doi.org/10.1080/23746149.2016.1259585

[7] Hu, Miao et al. "Memristor Crossbar-Based Neuromorphic Computing System: A Case Study", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, 2014,.pp. 1864-1878. DOI: https://doi.org/10.1109/TNNLS.2013.2296777

[8] S. Agarwal, O. D. Parekh, T. Quach, C. D. James, J. B. Aimone and M. Marinella,

”The energy scaling advantages of RRAM crossbars”, *2015 Fourth Berkeley Symposium on Energy Efficient Electronic Systems (E3S)*, Berkeley, CA, 2015, pp. 1-3. DOI: https://doi.org/10.1109/E3S.2015.7336818

[9] Raqibul Hasan, Tarek M. Taha and Chris Yakopcic, “On-chip training of memristor crossbar based multi-layer neural networks”, *Microelectronics Journal*, Volume 66, August 2017, Pages 31-40. DOI: https://doi.org/10.1016/j.mejo.2017.05.005

[10] M.S. Tarkov, “Mapping weight matrix of a neural network’s layer onto memristor crossbar”, *Opt. Mem. Neural Networks*, vol 24, 2015, p/ 109. DOI: https://doi.org/10.3103/S1060992X15020125

[11] M. A. Zidan, Y. Jeong, J. H. Shin, C. Du, Z. Zhang and W. D. Lu, ”Field-Programmable Crossbar Array (FPCA) for Reconfigurable Computing”, *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, Oct.-Dec. 2018, pp. 698-710. DOI: https://doi.org/10.1109/TMSCS.2017.2721160

[12] Chua, L. O, ”Memristor—the missing circuit element”, *IEEE Transactions on Circuit Theory*, Volume: 18 , Issue: 5 , September 1971, Page(s): 507 - 519. DOI: https://doi.org/10.1109/TCT.1971.1083337

[13] Strukov, D. B., Snider, G., Stewart, D. & Williams, R. S, ”The missing memristor found”, *Nature* 453, 80–83 (2008). DOI: https://doi.org/10.1038/nature06932

[14] Liu, T. et al., ”A 130.7 mm2 2-layer 32 Gb ReRAM memory device in 24 nm technology”, *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, San Francisco, CA, 2013, pp. 210-211. DOI: https://doi.org/10.1109/ISSCC.2013.6487703

[15] Prezioso, M. et al., ”Training and operation of an integrated neuromorphic network based on metal-oxide memristors”, *Nature* 521, 61–64 (2015). DOI: https://doi.org/10.1038/nature14441

[16] Yao, P. et al., ”Face classification using electronic synapses”, *Nat Commun* 8, 15199

(2017). DOI: https://doi.org/10.1038/ncomms15199

[17] Sheridan, P. M. et al., "Sparse coding with memristor networks", *Nat Nanotech* 12, 784–789 (2017). DOI: https://doi.org/10.1038/nnano.2017.83

[18] Li, C. et al., "Analogue signal and image processing with large memristor crossbars", *Nat Electron* 1, 52–59 (2018). DOI: https://doi.org/10.1038/s41928-017-0002-z

[19] Kim, S.; Kim, H.; Hwang, S.; Kim, M.H.; Chang, Y.F.; Park, B.G., "Analog Synaptic Behavior of a Silicon Nitride Memristor", *ACS Appl. Mater. Interfaces* 2017, 9, 40420–40427. DOI: https://doi.org/10.1021/acsami.7b11191

[20] Hsieh, C.C.; Roy, A.; Chang, Y.F.; Shahrjerdi, D.; Banerjee, S.K., "A sub-1-volt analog metal oxide memristive-based synaptic device with large conductance change for energy-efficient spike-based computing systems", *Appl. Phys. Lett.* 2016, 109, 223501. DOI: https://doi.org/10.1063/1.4971188

[21] Chang, Y.F.; Fowler, B.; Chen, Y.C.; Zhou, F.; Pan, C.H.; Chang, T.C.; Lee, J.C. "Demonstration of synaptic behaviors and resistive switching characterizations by proton exchange reactions in silicon oxide", *Sci. Rep.* 2016, 6, 21268. DOI: https://doi.org/10.1038/srep21268

[22] Hu, Miao & Li, Hai & wu, Qing & Rose, Garrett,"Hardware realization of BSB recall function using memristor crossbar arrays", *Proceedings - Design Automation Conference*, 2012. DOI: https://doi.org/10.1145/2228360.2228448

[23] Warren S. McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", *Neurocomputing: foundations of research*, James A. Anderson and Edward Rosenfeld (Eds.). MIT Press, Cambridge, MA, USA 15-27, 1988. DOI: https://doi.org/10.1007/BF02478259

[24] De Oliveira, Rodrigo & Fernandes Araújo, Ramon Cristian & Barros, Fabrício & Segundo, Adriano & Zampolo, Ronaldo & Fonseca, Wellington & Dmitriev, Victor & Brasil, Fernando, "A System Based on Artificial Neural Networks for Automatic

Classification of Hydro-generator Stator Windings Partial Discharges", *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, 16. 628-645, 2017. DOI: http://dx.doi.org/10.1590/2179-10742017v16i3854

[25] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks", *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Volume 1, Pages 1097-1105, Lake Tahoe, Nevada — December 03 - 06, 2012. DOI: https://doi.org/10.1145/3065386

[26] Gu, Jiuxiang; Wang, Zhenhua; Kuen, Jason; Ma, Lianyang; Shahroudy, Amir; Shuai, Bing; Liu, Ting; Wang, Xingxing; Wang, Li; Wang, Gang; Cai, Jianfei; and Chen, Tsuhan, "Recent Advances in Convolutional Neural Networks". https://arxiv.org/abs/1512.07108v6

[27] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang, "Deep Learning for Extreme Multi-label Text Classification", *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*, ACM, New York, NY, USA, 115-124, 2017. DOI: https://doi.org/10.1145/3077136.3080834

[28] Leibin Ni, Hantao Huang, Zichuan Liu, Rajiv V. Joshi, and Hao Yu. 2017,"Distributed In-Memory Computing on Binary RRAM Crossbar", *J. Emerg. Technol. Comput. Syst.*, Vol. 13, 3, Article 36, March 2017, 18 pages. DOI: https://doi.org/10.1145/2996192

[29] Zhang, Leqi & Cosemans, Stefan & Wouters, Dirk & Groeseneken, Guido & Jurczak, M. & Govoreanu, Bogdan, "On the Optimal ON/OFF Resistance Ratio for Resistive Switching Element in One-Selector One-Resistor Crosspoint Arrays", *IEEE Electron Device Letters*, 36. 1-1, 2015. DOI: https://doi.org/10.1109/LED.2015.2427313

[30] L. Zhang, S. Cosemans, D. J. Wouters, G. Groeseneken, M. Jurczak and B. Govoreanu, "Selector design considerations and requirements for 1 SIR RRAM crossbar array," *2014*

IEEE 6th International Memory Workshop (IMW), Taipei, 2014, pp. 1-4. DOI: https://doi.org/10.1109/IMW.2014.6849358

[31] Khaled Nasr, Pooja Viswanathan and Andreas Nieder, "Number detectors spontaneously emerge in a deep neural network designed for visual object recognition", *Science Advances*, 08 May 2019: Vol. 5, no. 5, eaav7903. DOI: https://doi.org/10.1126/sciadv.aav7903

[32] Jouppi, Norman P.; Young, Cliff; Patil, Nishant; Patterson, David; Agrawal, Gaurav; Bajwa, Raminder; Bates, Sarah; Bhatia, Suresh; Boden, Nan; Borchers, Al; Boyle, Rick; Cantin, Pierre-luc; Chao, Clifford; Clark, Chris; Coriell, Jeremy; Daley, Mike; Dau, Matt; Dean, Jeffrey; Gelb, Ben; Ghaemmaghami, Tara Vazir; Gottipati, Rajendra; Gulland, William; Hagmann, Robert; Ho, C. Richard; Hogberg, Doug; Hu, John; Hundt, Robert; Hurt, Dan; Ibarz, Julian; Jaffey, Aaron; Jaworski, Alek; Kaplan, Alexander; Khaitan, Harshit; Koch, Andy; Kumar, Naveen; Lacy, Steve; Laudon, James; Law, James; Le, Diemthu; Leary, Chris; Liu, Zhuyuan; Lucke, Kyle; Lundin, Alan; MacKean, Gordon; Maggiore, Adriana; Mahony, Maire; Miller, Kieran; Nagarajan, Rahul; Narayanaswami, Ravi; Ni, Ray; Nix, Kathy; Norrie, Thomas; Omernick, Mark; Penukonda, Narayana; Phelps, Andy; Ross, Jonathan; Ross, Matt; Salek, Amir; Samadiani, Emad; Severn, Chris; Sizikov, Gregory; Snelham, Matthew; Souter, Jed; Steinberg, Dan; Swing, Andy; Tan, Mercedes; Thorson, Gregory; Tian, Bo; Toma, Horia; Tuttle, Erick; Vasudevan, Vijay; Walter, Richard; Wang, Walter; Wilcox, Eric; Yoon, Doe Hyun, "In-Datacenter Performance Analysis of a Tensor Processing Unit™", *Proceedings of the 44th Annual International Symposium on Computer Architecture*, Pages 1-12, Toronto, ON, Canada — June 24 - 28, 2017. DOI: https://doi.org/10.1145/3079856.3080246

[33] Qiao, Yu (2007). "THE MNIST DATABASE of handwritten digits". Retrieved 18 August 2013.

[34] Platt, John C. (1999). "Using analytic QP and sparseness to speed training of support vector machines", *Advances in Neural Information Processing Systems: 557–563*,

Retrieved 18 August 2013.

[35] Alejandro Baldominos, Yago Saez and Pedro Isasi, "A Survey of Handwritten Character Recognition with MNIST and EMNIST", *Appl. Sci.*, 2019, 9(15), 3169. DOI: https://doi.org/10.3390/app9153169

[36] Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", 2009. https://www.cs.toronto.edu/~kriz/cifar.html

[37] Mu, Yadong & Chen, Xiangyu & Liu, Xianglong & Chua, Tat-Seng & Yan, Shuicheng, "Multimedia semantics-aware query-adaptive hashing with bits reconfigurability", *International Journal of Multimedia Information Retrieval*, Volume 1, Issue 1, April 2012, pp 59–70. DOI: https://doi.org/10.1007/s13735-012-0003-7

[38] Chen, Chen & Ren, Yuzhuo & Kuo, C.C. (2016), "Global-Attributes Assisted Outdoor Scene Geometric Labeling", *Big Visual Data Analysis*, SpringerBriefs in Electrical and Computer Engineering, Springer Singapore, pp 93-120, February 2016. DOI: https://doi.org/10.1007/978-981-10-0631-9_5

[39] https://machinelearningmedium.com/2017/09/08/overfitting-and-regularization/

[40] Aurélien Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", O'Reilly Media, 2017.

[41] Sergey Zagoruyko and Nikos Komodakis, "DiracNets: Training Very Deep Neural Networks Without Skip-Connections", 2017. Image Courtesy of Jae-sun Seo. DOI: https://arxiv.org/abs/1706.00388

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition", 2015. https://arxiv.org/abs/1512.03385

[43] Qizhe Xie, Eduard Hovy, Minh-Thang Luong and Quoc V. Le, "Self-training with Noisy Student improves ImageNet classification", 2019. https://arXiv:1911.04252[cs.LG]